

A Handbook on DIY Electronic Security and Espionage



Luka Matic



VISIT...

LANZAROTE
Caliente.COM

A Handbook on DIY Electronic Security and Espionage



Luka Matic



an Elektor Publication

● This is an Elektor Publication. Elektor is the media brand of
Elektor International Media B.V.

78 York Street

London W1H 1DP, UK

Phone: (+44) (0)20 7692 8344

© Elektor International Media BV 2021

First published in the United Kingdom 2021

● All rights reserved. No part of this book may be reproduced in any material form, including photocopying, or storing in any medium by electronic means and whether or not transiently or incidentally to some other use of this publication, without the written permission of the copyright holder except in accordance with the provisions of the Copyright, Designs and Patents Act 1988 or under the terms of a licence issued by the Copyright Licensing Agency Ltd, 90 Tottenham Court Road, London, England W1P 9HE. Applications for the copyright holder's written permission to reproduce any part of this publication should be addressed to the publishers. The publishers have used their best efforts in ensuring the correctness of the information contained in this book. They do not assume, and hereby disclaim, any liability to any party for any loss or damage caused by errors or omissions in this book, whether such errors or omissions result from negligence, accident or any other cause.

● British Library Cataloguing in Publication Data

Catalogue record for this book is available from the British Library

● ISBN: 978-3-89576-465-3

● EISBN: 978-3-89576-466-0

Prepress production: DMC | dave@daverid.com

Printed in the Netherlands by Ipskamp



Elektor is part of EIM, the world's leading source of essential technical information and electronics products for pro engineers, electronics designers, and the companies seeking to engage them. Each day, our international team develops and delivers high-quality content - via a variety of media channels (e.g., magazines, video, digital media, and social media) in several languages - relating to electronics design and DIY electronics. www.elektor.com

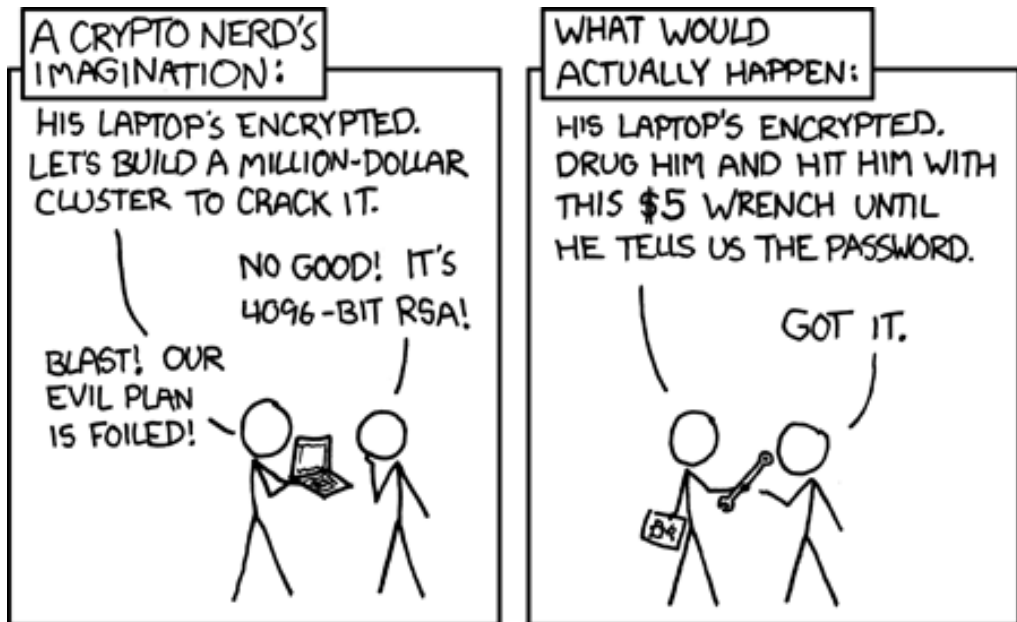
FOR ALICE AND BOB:

Stay vigilant.

Speed kills.

Trust no 1.

READY? ■



The author wishes to express his sincere thanks to XKCD (xkcd.com) for providing permission to include comic 538 in this book.

Table of Contents

Chapter 1 • All security problems solved perfectly - or perhaps not?	10
1.1 • Popular misconceptions.	11
1.1.1 • (Mis)understanding the basic principles of security.	11
1.1.2 • Why design something new?	12
1.1.3 • Moore's law and its corollary on security	14
1.1.4 • Espionage in the past and present.	14
1.2 • Omnipresent, unrecognised, and unaddressed problems.	17
1.2.1 • Liability problem	17
1.2.2 • Failure to recognise important problems to solve	18
1.2.3 • Black box problem: Why should I care HOW my super-gizmo gets its work done?	22
1.2.4 • Reluctance to properly address the "impossible" scenarios	22
1.2.5 • The problems that electronic engineers can't solve.	24
1.3 • Low tech rules - very unintuitive	25
1.4 • My design philosophy and approach to security	28
Chapter 2 • Methods of Attack	31
2.1 • Methods to counteract	31
2.2 • Mathematical crypto-analysis.	32
2.2.1 • Brute-force	34
2.2.2 • Attacks on RNGs	36
2.3 • Buffer-overflow	37
2.3.1 • Types of buffer-overflow attacks	38
2.3.2 • Von Neumann's vs. Harvard architecture	39
2.4 • Side-channel attacks	40
2.4.1 • TEMPEST - a type of side-channel	42
2.4.2 • How to defend on a DIY budget?	49
2.5 • Hardware Trojans	54
2.5.1 • Types of hardware trojan	55
2.5.2 • East German Z80 clone vs. the newest 10nm FPGA	56
2.5.3 • Planting, detecting, and countermeasures	58
2.6 • Exploiting inherently insecure physical properties	61
2.6.1 • Deleting HDD and SSD devices	62
2.6.2 • Recovering data from old (E)EPROMs.	63

2.6.3 • SRAM and DRAM data remanence	68
2.6.4 • Cold boot attacks	72
2.6.5 • What can we do DIY?	74
Chapter 3 • Random Number Generators	78
3.1 • A good RNG as a necessary link in the security chain	78
3.1.1 • Defining specs for a good RNG for use in a crypto system	78
3.1.2 • NIST testing	79
3.1.3 • Other ways to use NIST RNG tests for security evaluation	81
3.2 • Types of RNGs available today and possible problems	82
3.2.1 • Pseudo-random numbers generators (PRNG)	82
3.2.2 • Highly integrated TRNGs	84
3.2.3 • Black-box TRNGs	84
3.3 • Elektor TRNG solves some problems, but...	85
Chapter 4 • Cryptography on paper, computer, and in the real world	90
4.1 • Why do cryptosystems fail?	90
4.1.1 • The famous ENIGMA	90
4.1.2 • VENONA affair	91
4.1.3 • Mathematics is perfect - well almost...	94
4.1.4 • Humans are definitely not perfect	95
4.2 • More problems and more misconceptions	96
4.2.1 • Loose definitions	96
4.2.1.1 • Let's try to define encryption strength...	98
4.2.1.2 • What is encryption, and what is not?	99
4.2.2 • Symmetric and asymmetric encryption	101
4.2.3 • PGP affair	103
4.2.4 • Quantum computers	105
4.2.5 • Reversing an implication and T-com payphones	106
4.3 • Black-box cryptography	110
4.3.1 • "Crypto AG" affairs	110
4.4 • Elektor OTP Crypto Shield	112
4.4.1 • Key distribution problems	117
4.5 • Tamper-evident Box solves some problems, but...	119
Chapter 5 • A few more cheap and highly secure gadgets	124
5.1 • SD card-to-SD card copier	124

5.2 • SD card-to-Cassette tape copier	126
5.3 • ZMC80 system by Lee Alan Hart	131
5.3.1 • Crypto development shield add-on	135
5.3.2 • Buffer-overflow protection on hardware level	138
5.3.3 • Stack smashing and code obfuscation	140
5.4 • Mg-flash analogue memory for Tamper-evident Box	141
5.5 • Security by obscurity	145
5.6 • MyNOR CPU-less computer by Dennis Kuschel	146
Chapter 6 • Hands-on!	150
6.1 • TEMPEST attack demos	150
6.1.1 • TEMPEST on a dot-matrix printer	150
6.1.2 • TEMPEST on a PS/2 or an USB keyboard	155
6.2 • Buffer-overflow attack demos	158
6.2.1 • Smashing the stack on ZMC- Z80	162
6.2.2 • Injecting and executing an arbitrary code	164
6.3 • SRAM burnt-in data recovery	170
6.4 • Cold-boot attack demo	178
Chapter 7 • A few more ideas to work on	182
7.1 • SIGSALY-2 “Reloaded”	182
7.2 • Microwave oven - an innocuous machine?	188
7.3 • “Funcard” system for secure digital signing and decryption	191
7.4 • TEMPEST-proof terminal	200
7.5 • False Morse signature generator	201
7.6 • Encrypted ROMs	205
7.7 • Asynchronous computers	208
7.8 • DIY device-a supervisor for a “suspicious” commercial device	211
• Conclusion	215
• References	216
• Index	220

Chapter 1 • All security problems solved perfectly - or perhaps not?

Here we are, in the 21st century, and the first 20 years have passed. All the electronics are cheap, fast, digital, highly reliable, highly integrated, and easily obtainable. Electronic components (once upon a time - I still remember) that needed extensive paper-catalogue shuffling, walking shop-to-shop and phone-call searching, and even smuggling across the Cold War borders, are now only a few clicks away. Books are relatively cheap, and fast internet is here for almost everyone - which means all information and knowledge required to design almost anything is also accessible to everyone interested.

On the other hand, **the more technology advances, the less secure it becomes**. This may not be so obvious as the facts in the previous paragraph, but it is exactly why I decided to write this book - I wanted to properly address routinely overlooked security issues of modern hi-tech. Let's first consider some other, also not so apparent facts about electronic technology nowadays.

Almost every electronic device we could have dreamt about (once upon that time) is now a few clicks away. Back then it was obvious that we needed faster computers, higher integration, better display resolution, more memory, faster refresh rates, lower prices, etc. Some technologies have already reached perfection (for all practical purposes) but are still being improved (for some unknown reason) - I am still thinking about buying the newest 800Hz wide-screen OLED, for Toxy, my feline lab assistant. We often watch spy movies together, so maybe he could inform me about the enhanced visual experience (only falcons and cats can process a video signal that fast in real-time) since the most perfect human eyes (like a fighter pilot's) can't notice any improvement above 150Hz.

This book is not about this sort of technology. Such discussion would quickly get boring, so I won't waste your time. Other technologies were suddenly shown to be extremely expensive and overrated hypes - **like invisible stealth aircraft** (almost always advertised as **invisible**, since the 1940s, in the strictest sense of this word, meaning **invisible** at any distance, for any radar frequency, mainly to justify enormous amounts of money spent). In theory, you can expose it yourself by carefully reading the first 50-100 pages of a basic microwave electronics textbook or any decent book on the basics of aerodynamics. In practice, after 50+ years and a probably still never-ending, multi-trillion waste of taxpayers' money was finally debunked in March 1999 with "obsolete" 1960s microwave RF and automation technology revamped by a small group of engineers operating on a shoestring budget. Needless to say, they started their remarkable project by reading their textbooks!

I have nothing original to write about this sort of technology, so I won't bother you with this. I will just remind you of the moral of the whole story - **please read your books carefully**. Don't rush to slap up something too quickly, don't do it to make money, and take your time to decide which project is worth working on in the first place. Don't be intimidated just because your opponent has more money and resources. Rely on your knowledge more than on expensive advertising as much as you can.

Considering the state of art today, it took me quite some time to decide what to work on. What is truly essential? Not available cheap-from-China? Not reliable enough? Above all this, what can be designed and assembled in an average home lab?

The answer is in this book. Simple, reliable, and secure encryption devices fully fit the aforementioned criteria! First, I will explain the basic problems of electronic security and present my solutions. I will try to relate the problems to real-world historical examples. You will see that for every problem solved, a few more also worth solving emerge. Therefore there are lots to work on.

1.1 • Popular misconceptions

1.1.1 • (Mis)understanding the basic principles of security

If many electronic devices have already reached the point of perfection, where any further development makes no practical sense, how does this reflect on electronic security and espionage? Hi-tech helps to spy on someone more discreetly and more effectively every millisecond of 24 hours a day - this is very easy to understand, but wait a minute... what happens **the other way around**? This is maybe not so obvious, but the situation is that a high-tech spy can be even more easily **spied upon**, or to put it in other words, the defence against spying becomes more and more complicated. Defending a modern complex system (like any PC or smartphone) is extremely complicated, if not impossible.

Let's first introduce a standard set of "crypto-characters" that are used in the security analysis of various hypothetic scenarios. Alice and Bob are two "good" spies trying to communicate and keep their comms secret. "Evil" Eve the eavesdropper is a passive spy who will try listening to their communications, cracking their codes, following them, dumpster-diving¹, or anything else without actively interfering with them. "Malicious" Mallory is an active spy who will actively interfere - by planting false messages, picking locks, breaking and entering, kidnapping, planting listening devices (bugs), planting viruses, planting pre-rigged electronic components to Alice's workshop, etc... These are four basic characters, and we may add a few more later, like **Trudy** the intruder, and **Walter** the warden.

It is much easier for Eve and Mallory to penetrate Alice's general-purpose PC - they need to find only one weak spot (a general-purpose PC has **many**, trust me), while Alice needs to defend them all simultaneously. Furthermore, she needs **to identify** all weak spots first (to prepare for any kind of effective defence), and this is even harder. This would first require Alice to be familiar with every piece of hardware and software used in her PC and how many lines of machine code Windows 10 OS has to begin with. How many top-class engineers do you know who are proficient with hardware and software, and also with the mathematics of cryptography? With analogue and digital electronics and then with machine code of their PC's particular CPU, and then comes the security...

Security is a chain that is as **strong as its weakest link**, which means that more complex

1 dumpster-diving: collecting information from physical items found in Alice's trash-can.

systems have many more potential links to rip - only one is enough, and evil Eve wins. A typical example of misunderstanding this basic principle of security is the PGP affair (we will address this later in more detail) from the 1990s. Phil Zimmermann did an exceptional job, of course, by programming the **PGP**² software, to effectively use the **RSA**³ public-key encryption and provide good crypto protection to everybody. The real problem that arose is that many people started thinking that **good cryptography can solve all their security problems**. The mathematics of cryptography is perfect: RSA and El-Gamal can't be cracked in any reasonable time if properly configured, the PGP program that implements it on a PC can have a bug here and there, but even if perfectly debugged... it still runs on a general-purpose PC, right? Why should Eve waste her time trying to brute-force Alice's private key, or crypto-analysing the RSA, when it is much easier to send Mallory to plant a key-logger, or a screenshot-capture Trojan on Alice's PC? Or even to eavesdrop on a so-called **TEMPEST**⁴ radiation from Alice's keyboard or monitor from across the street? Wim van Eck did it from several hundred meters away in 1980s. Why not record snapshots of Alice's PC's RAM when it multiplies large prime numbers for generating key pairs? Or install a hidden camera to record plaintext on her screen? This paranoid list goes on and on, and I have barely started to analyse it properly...

As we can now see, to fully secure a general-purpose PC or smartphone requires extensive knowledge and painstaking work - i.e. a highly paranoid analysis of every possible weak link in the chain - PGP itself is not enough. Very few top-class engineers have all the knowledge, and even fewer have time and patience to work this all out thoroughly. And yet, many more people need good security and privacy!

Does this mean that our poor, low-budget spies Alice and Bob should abort their operations and give up? Can they do anything DIY at all? Maybe they should start looking for a job in a major 3-letter agency... But wait, big companies have large and complex systems and consequently leak secret data even easier than a single Alice's or Bob's PC, which is already too complex itself!

The good news for Alice and Bob is they **can do a lot on a low DIY budget** - they just have to take a different approach! This is what my book is about. **My main idea is to help Alice and Bob**. I will explain it step-by-step in the following chapters. It will get more interesting, I promise!

1.1.2 • Why design something new?

This has unfortunately become a very common rhetorical question, even within our community of electronic engineers. Everything is already improved up to the point of

2 PGP: Pretty Good Privacy.

3 RSA, besides El-Gamal, is the most important asymmetric encryption method-using a pair of public and private keys. RSA is based on quick and easy multiplication of two big prime numbers and slow and difficult factorisation of their product.

4 TEMPEST: residual signals inevitably generated and transmitted by all the electronic and even purely mechanical devices. Some consider the name an acronym, while others disagree. It poses a significant security risk. It will be analysed in more detail in chapter 2.

ultimate perfection anyway, so why bother. Maintenance, troubleshooting and repairing of the electronics on existing machinery brings decent money, so why bother with design engineering; it is too difficult, expensive, and time-consuming, and carries too much liability. Maybe just because some problem is there, so let me just try to solve it my way - will make at least a good practical exercise, just to boost my confidence.

After just a little thinking, in the case of electronic security and cryptography, it becomes apparent that the situation in this branch is quite opposite! Good cryptography software (like PGP) is easily available to everyone. Good anti-virus software (including all kinds of software protection, i.e. anti-spyware, anti-phishing, firewalls, etc.) is also easily available, but...

OK, but what about secure hardware? Somebody, please e-mail me a link to some online store to cheaply buy (preferably all in one place, to avoid too much clicking) a good true random number generator, a strong encryption/decryption device, a secure keyboard and monitor, a secure printer, a secure RF burst transmitter/receiver, a tamper-proof storage box, a CPU that I can trust 100% (i.e. 100% certified not pre-rigged with a hardware Trojan and with its silicon die microcircuit blueprints available), **zeroisable**⁵ RAM, EPROM, HDD and SSD, a secure device to copy sensitive data between different memory media, reliable thin paper and invisible ink, a secure device to handle bitcoin payments... Most important of all, besides functionality, I want all devices to be fully documented, open-hardware and open-software (preferably with CPU single-stepper option pre-installed), implemented on PCBs with thick enough traces, all test points installed and fully accessible, **so I can check every analogue and digital signal and monitor every detail of every device's full operation in real-time**. The last sentence is my most important requirement (without it I can't trust the equipment to handle my sensitive secrets, sorry) - maybe you will find some well-advertised, very expensive, and overhyped stuff (like crypto-phones or stealth aircraft - remember?), but I am pretty sure that none will fulfil that crucial requirement, the most important one, *conditio-sine-qua-non*.

Checking a device's functionality is easy, but testing its security is very difficult, especially if it is a black box. If Alice can't test and check every aspect and scenario of its operation, it is simply not trustworthy - this is the way things are and it is regularly overlooked.

Most of the aforementioned hardware simply doesn't exist. It is much easier to design and assemble something trustworthy yourself than analyse another engineer's design. This is one very important fact, self-explained, but also vastly overlooked. This is simply the way things are. Many more devices that haven't come to my mind yet are also needed, but surely not readily available off-the-shelf.

This is exactly why it makes sense to work hard on your own designs.

5 Zeroise: to effectively fully erase a medium without any possibility of data recovery.

1.1.3 • Moore's law and its corollary on security

Moore's law is a very well-known empirical fact - the number of transistors on the highest-integrated IC approximately doubles every year. This means that track width consequently decreases (from μm range in the 1970s to nm range today) and prices go down, while performance keeps increasing, practically everything in a favour of the customer.

There is also a less known Moore's second law which states that as prices for a customer decrease, prices for a producer (R&D, manufacturing, and testing) increase to fulfil Moore's first law.

So how does this reflect on the security of electronic hardware? Moore's second law may give you a hint - the expenses of **testing** a higher-integrated IC are getting higher. OK, but this refers only to testing of that IC's **functionality**, not **security**. If the cost of testing functionality of a more complex IC becomes higher, it unfortunately, means the costs of fully testing its security become exponentially higher! Why? Functionality test of an electronic device is a test to confirm only if its performance within **normal operating conditions** (which is a set of pre-defined specs) simply meets these specs. A security test is much more complicated -mainly because this is not a test within the scope of the pre-defined specs of some normal conditions. To test security, we need to think of a much wider scope of conditions. Attackers don't play by the rules, and definitely not within the range of any pre-defined specs. They are devious and unpredictable. Testing security requires many independent experts and lots of time. New designs (higher integration, narrower tracks) are more vulnerable than old ones that are robust and already tested and re-tested over a long period. Furthermore, integrating a device makes it less secure than physically splitting it into multiple ICs or modules - a highly integrated version will have much fewer accessible test points to test various signals and will also provide more opportunity to Mallory to plant hardware Trojans undetected.

Testing the security is much more complicated and hence more expensive than testing the functionality - this means that it requires extra financial investment that is rarely justified and never returned in a reasonable amount of time.

To sum it all up, as technology advances and the level of integration, speed, and overall performance increases, the level of security decreases even faster! We will analyse this in more detail later, as we continue to reveal many other pitfalls of modern highly integrated electronics. All modern trends of electronic technology progress tend to work against security! Seems a bit unintuitive? Well, there will be more to it - I have barely started...

1.1.4 • Espionage in the past and present

Now we will make a comparison between the level of technology and methods of espionage and counter-espionage available to Alice, Bob, Eve, and Mallory with a focus on the distinction between the 20th (and before that) and 21st century. We will see that advanced technology in the 21st century changed not only the technical aspects but also the basic methods and principles of espionage compared to the previous era.

I was born in Yugoslavia - a communist dictatorship, not so technologically advanced like the Soviet Union or East Germany. Unlike them, Yugoslavia was very short on highly educated and trained personnel in its 3-letter agencies. As a member of the **Non-Aligned Movement**⁶, it wasn't actively involved in the Cold War between East and West Bloc (or more precisely, Warsaw Pact and NATO) and the high-tech arms race. Consequently, it didn't need any high-tech espionage to keep its regime in power. Its 3-letter agencies dealt only with spying and intimidating Yugoslav citizens, using mostly very "low-tech" methods. With nothing of interest going on in my neighbourhood, I started reading books. Good spy movies were difficult to get. In 1985, when I got my first computer, a Spectrum ZX-48, smuggled from Trieste (Yugoslav civilians were not allowed to buy computers, because "an enemy might get a hold of them"), a crossword-puzzle magazine "Kviz" by "Vjesnik" from Zagreb started publishing a series of fictional spy-story puzzles under a very catchy name - **"A crash-course on clandestine diplomacy and espionage"**. They caught my attention immediately: the puzzles were extremely difficult to solve. I started buying every new issue of Kviz the day it arrived at the local kiosk, although it got me wondering how it slipped through communist censorship since this could also provide the "enemy" with a lot of useful knowledge, the same as it did for me.

Now heavily armed both with illegal high-tech from the West, and newly gained knowledge, I decided to try to do some cryptography on my Speccy. I made BASIC programs, first for **Caesar's cipher**, and then more advanced **Vigenere's cipher**, and then ran into problems with truly random numbers for **Vernam's one-time pads**⁷. To get ready for Morse Rx/Tx (any Cold War spy's long-range tool of trade), I made a program to convert ASCII text to Morse dit-dit-daah and the other way around. Then I hacked my Sinclair joystick to connect my Morse key to its fire button... At least I got my first nice taste of "forbidden" and learned what an "enemy" might use it for.

In 1989. The Berlin Wall went down, followed by Yugoslavia later in 1990, and we finally got first democratic elections, only to get another dictator (one of the last in Europe), until he died in 1999. At the time, I was 23 years old, a student at FER Zagreb, now with the experience of living under two different dictatorships and more technical knowledge, so I started thinking of ways and means to make some use of it.

And then came the 21st century: I started my cooperation with Elektor, and finally got to start solving real cryptography problems in practice (that I was thinking over and over for quite some time then) and publishing articles with working prototypes. New methods and threats appeared, requiring a new approach.

To start solving security problems, we need to properly identify the threats first. This is

6 NAM was a 3rd Bloc during the Cold War. Enabled a certain business cooperation among the 3rd World countries, independently of NATO or WP Bloc. Yugoslav president Tito, Jawaharlal Nehru of India and Fidel Castro of Cuba were the key persons. NAM actually survived the end of the Cold War, but its political influence nowadays is not significant.

7 OTP, or Vernam's cipher is the only method that can be mathematically proven as unbreakable, if properly used. Its key is a very long sequence of random numbers, as long as a plaintext message. Each random number is used only once, to encrypt one letter of the plaintext, usually by bitwise XOR-ing. All encryption methods will be explained later.

why it is important to make a clear distinction between 20th and 21st centuries. In the 20th century, technical resources were limited, which required more manpower. Consequently, it was possible to spy on and survey only a limited number of citizens. A tape recorder had to be hooked to your phone line, and an agent had to replace the tapes (they had limited capacity, up to several hours). A radio transmitter bug (a covert listening device) had to be placed hidden inside your office. Then some unfortunate agent had to listen to kilometres of tapes, just to conclude that your “discussions” with your wife, mistress, and mother-in-law contained nothing security-critical for the government. Cameras were still big, unwieldy, and expensive and hence only used in some special situations. Following a suspect on a street required physically tailing him on foot or by car, and such surveillance could be evaded. Most important of all, you had to be engaged in some suspicious activities first, before STASI⁸ would dispatch a team to follow you and record your conversations. This means that it wasn’t dangerous to tell a political joke over an analogue telephone, because the conversation was probably not monitored. One more important detail - civilian and military systems used different hardware, software, and communication channels. Real-time encryption of telephone communications was seldom used, mainly because of the too expensive hardware required. On the other hand, although getting the radio frequency assignment chart from the government was usually difficult, after re-tuning your AM or FM receiver out of their standard bands, it was possible to listen to e.g. police or air-force conversations because they were not encrypted in real-time. Most of the high-tech equipment was very expensive and hence not available to low-budget rogue spies.

Alice and Bob could safely use payphone-to-payphone calls to securely exchange encrypted messages. After finishing the call and wiping their fingerprints, they could safely leave. Many more payphones were in service than nowadays, and waiting for a call beside a payphone wasn’t considered suspicious behaviour - especially while cellphones were not yet available (or still too expensive). Landline connections were still very expensive in some areas; so many houses didn’t have a phone line installed. Payphones were not under 24/7 video surveillance - so Alice and Bob didn’t even have to bother wearing a disguise. Even if the conversation was taped, this meant nothing, because they wouldn’t use the same payphones twice. If intercepted by Eve - a human operator, a payphone-to-payphone conversation sounding something like **“Bravo-3-6-8-7, Oscar-5-0-9-1, Yankee-4-9-6-4, Lima-2-7-5-1...”** during the Cold War would immediately alert the counter-espionage, but the time needed to trace the call, locate both payphones and dispatch the police was surely more than 10 minutes- so Alice and Bob were safe as long as they kept their conversations shorter than 5 minutes. Btw, this is exactly why Yugoslav communists **disabled** payphone-to-payphone calls. An “enemy” could definitely have tried using them.

Now, in the 21st century, most of the aforementioned has changed. Cameras and microphones are everywhere, almost all online (some are not 24/7, but you usually don’t know which, where, and when) and networked. Every smartphone, PC, and even old GSM phones can easily become personal listening devices, by only uploading and activating a software Trojan. No need for Eve to waste time listening to bugs planted under your desk. Everything is recorded and kept. HDD storage space is almost unlimited, so Eve doesn’t

8 Ministerium für **STA**ats**SI**cherheit - East German intelligence until 1989. Not 3, but 5-letter agency known for their very high-tech methods, many invented by their own experts.

need to bother with connecting tape recorders and replacing tapes. Artificial intelligence can analyse the conversations and flag them if there is something suspicious, so it can be reviewed later by a human operator. No need for agents to follow you around, on foot or by car, since cameras are everywhere, and more and more of them are being installed. Your activities are recorded and kept even if you are not under any suspicion. Today, civilian and military systems often use the same hardware. A NATO officer's laptop may be a "Toughbook", but is still a general-purpose PC running under Windows OS. Almost all conversations are digitally encoded and encrypted (GSM, VOIP, different protocols), so just knowing the RF frequency of the GSM signal will not enable Eve to listen in. The prices of modern high-tech came down to the point where they became affordable for our low-budget spies.

I mentioned the payphone-to-payphone calls previously just to point out how drastically some security aspects have changed. Being one of the **safest** methods of the 20th century, this has now become one of the **least safe** methods! Now, with cheap cellphones available to almost everybody, Alice will need to have a very good excuse for standing by a CCTV-monitored payphone waiting for Bob's call. She will also need a very good disguise, actually several disguises ready to change quickly, and an even better escape plan, since she can expect every street in an average urban area to be at least partially monitored by at least one camera. The total number of operational payphones today is down to 5% of the maximum reached at the turn of the century, so Alice and Bob will have a hard time finding a new pair of payphones for each contact. Their conversation, now surely recorded and monitored by Eve the Artificial Intelligence will be immediately flagged, mainly because nowadays it is an extremely rare payphone-to-payphone call, and secondly due to unusually high percentage of numbers and spelled letters. Detection of location and reaction can be practically immediate, and evading the video surveillance after the call is very difficult.

With this in mind, it is important to get tuned to the rules and state-of-art of the 21st century - from my experience, I can tell that most people, even those born in the 21st century are still adopting a 20th century's mindset and this is not good. Alice and Bob must adapt their actions to the new rules, or they are lost.

1.2 • Omnipresent, unrecognised, and unaddressed problems

Proper identification of threats is crucial before we start to solve security-related problems. Otherwise, it is pointless. In this section, we will see how important it is, and unfortunately routinely overlooked.

1.2.1 • Liability problem

Maybe not the best one to start with, but the main reason is that almost nobody is aware of it. Let's start with a basic definition: an entry from the Oxford Dictionary, to be **liable** means to be held **legally responsible** for something.

This is relatively easy to understand in areas like e.g. civil engineering, shipbuilding, or electrical power engineering. A civil engineer is held liable if a bridge built according to

his drawings collapses, e.g. because he made bad structural calculations. A construction contractor is liable if he cuts corners and builds a cheaper bridge not according to design plans from the civil engineer. A facility manager is liable for refusing to evacuate a building that started to structurally fail hours before the main collapse (happened with the Sampoong Department Store in Seoul in 1995, along with lousy structural calculations and even worse construction job). A ship's captain is liable for the sinking of a ship because of disregarding safety procedures, etc.

Bad **safety** practice makes you liable for an accident, but what about bad **security**? Safety and security are two different things: I learned this in the 5 years I spent on board offshore construction vessels and oil drilling rigs. Safety means protection against **accidents**, while **security** refers to **deliberate attacks**.

There is much less liability in the world of security. I don't know much in detail about the legal side, but this is just a common practical fact. Police or private security guards are not liable for failing to protect you if you get wounded during a bank robbery. The police chief may be fired or demoted for failing to solve a string of bank robberies. Bad security guards can be sacked and replaced. My offshore construction company used to hire highly unreliable security for sailing through the Gulf of Aden (luckily, I never sailed on board one of those ships). They used to leave their loaded handguns everywhere around the ship, but they were much cheaper than professionals.

The situation in the world of electronic security (both hardware and software) is even worse. Vendors of bad security software (anti-virus, anti-spyware, firewall...) are **not liable** for any damage caused by malicious hacker attacks. Software security companies have another possibility that others don't have - if there is a security weakness exploited in ver 4.2, a patch will be uploaded to this company's website for everyone to download and upgrade the firewall to ver 4.3. They will consider the problem solved. Civil engineers or shipbuilders don't have this "luxury". Our ship sank due to structural failure, but ver 2.0 will be unsinkable, we promise. This is another reason why electronic security is often taken too lightly, by everybody involved - designers, vendors, and end-users.

Until something changes from the legal side, Alice and Bob will have to take more care about their security themselves, especially regarding hardware security - reliance on software security products is not enough. Anyway, they can't buy insurance against damage caused by leakage of their critical data to Eve, especially not if their operations are **illegal**.

1.2.2 • Failure to recognise important problems to solve

Before we start to design any security features, we first need to identify the real threats. The full scope of threat to defend against is seldom as obvious as it may seem. A simple example is wasting money and time installing a brand new 10-pin pick-proof lock on a plywood door, while also forgetting to put steel bars over a glass window in your basement. Why would anyone try to pick the expensive lock with mechanical pick-proof features when is it easier to slam on that weak door or even more discreetly break the basement window?

This is a very simple example, but the following are not that obvious:

1. Alice uses one-time pads, with perfect true random numbers sequences to communicate with Bob, so Eve can't crack the encryption. Alice uses the pen-and-paper method to encrypt messages before transmitting them by keying in manually the Morse code to a long-range shortwave radio transmitter. She prefers manual operation to prevent electronic leakage of data (more about this later). First, we will set it up in the Cold War stage. Let Alice be a Soviet spy in a safe place in Moscow. Bob is an illegal operating in the USA, and Eve is FBI. In this setup, Eve will have a very hard time trying to catch Bob. The communication is only one-way, and the encryption is unbreakable. Eve can try radio-locating Alice's transmitter (may be difficult at long distance, after several ionospheric reflections and refractions). Eve can tape Alice's transmissions as samples for later comparison. Bob can securely receive Alice's transmissions without drawing any attention from his neighbours - maybe he will suspend a few wires in his wooden attic to make a better shortwave antenna if the signal is weak. Eve could theoretically pick weak residual radiation coming from his **superhet**⁹ receiver's local LO-RF oscillator, only if she is very close in his neighbourhood- but he can also use a **TRF**¹⁰ receiver to mitigate this if he reaches that level of paranoia.

Now if Bob needs to transmit a message to Alice, things will get complicated. Eve still can't break his encryption, but the mere fact that he is transmitting an encrypted message may put him in trouble, after a successful triangulation of his position. This still doesn't flag him as a Russian spy, especially if he has a radio-amateur license - there is still no evidence that the message was sent to Alice, so he can get away with it.

The situation will get tough if Alice comes to the USA to operate as an illegal afterwards and starts transmitting. Eve will immediately know that she is a Russian spy after comparing her new transmissions with previously recorded samples sent from Russia. Every telegraph operator has a distinct rhythm of keying - like a signature or a fingerprint. The normal ratio of duration between dash and dot is 3:1, while a space between should be the same duration (1:1) as a dot. Some operators will have more like 2.8:1 for dash/dot ratio and 1.2:1 for dot/space. Besides these ratios, there are spaces between letters and words, and you get the operator's full signature. Since a Morse key also exhibits a button-bounce effect (like any mechanical switch), it is also

9 superheterodyne receiver: by mixing the RF signal received in antenna with a sinewave RF from a variable local oscillator (LO), a signal of the same baseband, but much lower carrier frequency (IF-or intermediate frequency equals a difference between LO frequency and received RF carrier frequency, typically 455kHz for 3-30MHz shortwave RF) is generated. Unlike TRF, the following amplifier stages need to be tuned to one fixed IF frequency only once, without the need to re-tune during operation. Tuning a superhet receiver to a transmitter station is done simply by varying the LO frequency.

10 tuned radio frequency receiver: no local oscillators, no IF. Every amplifier stage must be tuned and re-tuned to RF carrier repeatedly during operation. Variable RF tank circuits are much less selective than fixed-frequency IF filters. Low IF is easier to amplify than a much higher RF. TRF receiver is difficult to manually re-tune in operation (too many variables), but this may not be needed, if listening to the same RF frequency most of the time. Outperformed in any technical quality aspect by a superhet, except for the security - LO can always leak a few nW RF power to an Rx antenna. A good TRF has absolute zero RF leakage.

possible to identify the telegraph operator by dynamics of his hand's motion! With this in mind, Alice will have to quickly relocate and conceal her transmitter and antenna after each transmission.

The moral of this story is that even perfect encryption is not enough to secure your communications. Yet many people don't understand this - many thought PGP running on a PC would solve all the problems.

The threat of cracking the encryption in this example was solved perfectly, but if Alice and Bob didn't recognise the threat of radio-location and identification of operators by their Morse transmissions, they would lose.

This is why, besides encryption, RF engineers needed to invent burst-transmitters, spread-spectrum, frequency hopping, and other methods to conceal communication itself.

Besides the three threats in this scenario, perhaps there are even more threats from Eve and Mallory. Can you think of any?

2. With this in mind, it is easy to see that the same can happen if Alice and Bob try the same with an analogue modem on a classic telephone landline - the messages are encrypted, but the switching information for the telephone exchange is still sent in plain and therefore the call can be traced and located.
3. Even without encryption, many people think that if Alice and Bob buy cellphones for cash in a street, with a SIM card with top-ups also paid in cash that this will conceal their identity. Radio-location today is even faster, especially when aided by a smartphone's GPS, and analysis of contacts dialled can reveal the identity-especially if apart from the two of them they call other people, whose phones are registered. Above all, the identification of an individual human's voice is even easier than his Morse transmissions. Even if they decide to use some kind of voice distortion, they need to know that some distortion algorithms are reversible, and they, therefore, need to choose carefully one that reliably and irreversibly removes the redundant information from the human voice signal.
4. Telephone companies, both landline, and mobile invested a lot to prevent theft of their services. Even then it was not 100% effective (remember the Blue-box, Red-box, cracked stored-value payphone cards...) The point is that theft of cheap phone calls is a much lesser problem than the theft of identity and making phone calls with the stolen identity of legit users. Much less has been invested to solve this problem.
5. Sometimes, solving one security problem will cause another more dangerous one to arise - **but you may be unaware of it**. Car alarms became popular in the 1990s, mainly due to them becoming more affordable. **Car theft** was increasing steadily during that time, while I studied on FER. What car owners failed to realise is that investing to mitigate a car theft, even if 100% possible with a perfect car alarm immediately

puts them in danger of **carjacking**¹¹, which is a way more dangerous attack. Some carjackers in Zagreb resorted to killing drivers (!) instead of forcing them to drive at a gunpoint, which they considered less practical. Then one company started advertising footswitch-operated **flamethrowers** as protection against carjackers. I solved this problem by buying a 1979. Citroen GS for 1000 DEM. It served well until the end of my studies, (both for driving and learning about non-electronic hydraulic-pneumatic control loops on a piece of practical machinery - I was at the automation branch) and no decent car thief would ever try to steal it. This is another example of what can happen because of poor threat identification.

6. Now it's your turn. Try to analyse the possible problems for Alice and Bob if they use newspaper ads to secretly communicate. This method was allegedly popular during the Cold War, although I don't know of any confirmed real-life instances. Anyway, the overall concept seems very secure and difficult to detect. It goes something like this: Alice places an ad in a newspaper with a message for Bob. Some of the ad's text is meaningless, but some part contains an encrypted message (concealed to look like a normal plain text). Bob buys the newspaper (in paper form) at a local kiosk, then reads the ads without drawing any suspicion whatsoever. He knows how to recognise a message from Alice - to all other 100.000 people who bought the newspaper today (including Eve and Mallory) it looks just like an ad to sell an old second-hand Ford. No traces left.

Movies have been made about some "Eves" of the Cold War who had become maniacally obsessed trying to decode non-existent concealed messages in newspapers' advertisements (and crossword puzzles as well).

Now, to point out another important problem - encrypting a message is one problem, concealing it (hiding the fact that an encrypted message is what is being sent) is another problem. As we have just seen previously, concealing Alice's and Bob's true identities, and/or the mere fact that they are communicating is yet a third problem. Besides our usual set of four crypto-characters (**Alice and Bob**, trying to secretly communicate, **Eve** eavesdropper, passive spy, **Mallory** malicious active spy) we need to add **Walter a warden, guarding Alice and Bob**. He may be a prison warden if Alice is in his jail and still wants to communicate with Bob. It may also be a 3-letter agency of a repressive dictatorship. In this case, all Alice's messages are checked by Walter. He doesn't bother decrypting them (Eve will try to do this part), his job is to check Alice's messages for possible concealed ciphertext. Alice will not do well if her message looks something like "143-231-525-BV, 45-834-56-AF, ..." -Walter will immediately notice something suspicious. If she tries writing a normal-looking love letter to Bob, where e.g. every first letter in every word is a character of a ciphertext, then Walter will probably not notice anything suspicious and pass it on. This is called **steganography**: hiding secret messages in other messages.

11 Carjacking: hijacking a car, usually along with a driver, typically at a traffic light or parking lot.

In the case of secret messages in newspaper ads, the newspaper's editorial staff takes the role of Walter - they will certainly not publish a meaningless sequence of numbers as an ad. If Alice puts a non-existent 10-digit phone number as a contact (which is an encrypted message to Bob), this will probably be passed off as a typo. On the other hand, if Walter is on high alert, he may request the dialling and checking every phone number...

Your turn to continue this analysis! Are there any possible dangers for Alice and Bob? What methods of concealing the messages can they use? What are the possible traces they could leave? What is better from a security standpoint- printed newspaper or internet ads? Try to identify as many threats as possible!

1.2.3 • Black box problem: Why should I care HOW my super-gizmo gets its work done?

Well, you should. As long as it does what it is supposed to do, nobody cares. This is OK for most of the devices you use today - because we are mostly concerned about functionality, not security. If it fails to work, you can claim a warranty and request a new item. But if it leaks your critical secret, you can do nothing - you should have thought of security and done something beforehand.

I will address the black box problem again in more detail several times in the following chapters; I must start here because it is increasingly **omnipresent**, regularly **unrecognised**, and almost always **unaddressed**.

If it is a black box, you can't analyse its security- this is the main problem. Furthermore, opening a black box device will surely cancel any warranty, put you in a breach of a contract, and may even be illegal. One of the best solutions is to design something new, open-source, so it can be tested, and constantly improved and contributed to by other independent experts.

1.2.4 • Reluctance to properly address the "impossible" scenarios

This may look like another case under sub-section 1.2.2, but it isn't. In this case, Alice is aware of a possible threat. She properly recognises it, but doesn't point it to Bob because she thinks it is almost impossible, or because she thinks Bob would consider it ridiculous. Therefore, the threat is disregarded, and the consequences may follow.

The point is that many dangerous scenarios in this world are "not impossible- just highly improbable" (my favourite line from "The Hitchhiker's guide to the Galaxy"). Let's analyse a few examples and see in which way the highly improbable threats were considered:

1. During the Cold War, the actual invasion of West Europe by the Soviet army (and other members of the Warsaw Pact) was considered improbable, but still possible. The scenario of full occupation and collapse of West Europe, especially along with Great Britain (hasn't been invaded for almost 1000 years), where only small guerrilla armies are left to fight, sounds impossible today - especially for those of you born after

the Cold War. In the 1980s it became obvious that the Warsaw Pact started facing serious economical problems, although they were still keeping up with the West in high-tech engineering and arms race. Some NATO analysts considered though that the Warsaw Pact armies were in much better combat readiness at that time than NATO. Consequently, some of them concluded that WP might attack first, to use this temporary advantage before their communist economy collapsed.

Following this line of logic, to prepare for this worst-case scenario, the **FS-5000 Harpoon project** was started and finished with a fully functional portable long-range radio-communication system. It was designed and produced by well renowned TELEFUNKEN AG, not by some crazy conspiracy theorist, basement-dwelling social outcasts.

The design requirements were set very high:

- several thousand kilometres ranges, **without any land-based or satellite repeaters**
- to be operable by unskilled operators i.e the guerrillas fighting in occupied West Europe
- the size of a briefcase, battery-powered, lightweight, with portable antenna

Sounds “impossible”? Scenario or design requirements? Or both? Well, I know very few RF engineers nowadays who could tackle this, and I know even fewer engineers who would ever try to follow this line of security analysis.

2. Some people working in very tall skyscrapers allegedly keep **parachutes** in their offices. This idea is of course ridiculed by the vast majority, even after 9/11. More than 200 serious skyscraper fire incidents have been reported throughout the world, often with fatal consequences. Big skyscrapers are several hundred meters tall, and successful parachute jumps have been reported from heights as low as 30 meters. A fire brigade chief once remarked to a civil engineer designing a 200-meter tall skyscraper - “Excuse me, sir, there is no safe way to fight a fire above the 10th floor - and yet you keep building them higher and higher”. Is a skyscraper fire (and also any other skyscraper disaster) a scenario so “impossible” that a parachute and a parachuting course are a waste of money?
3. Small knives and many other innocuous objects are strictly not allowed to be carried onboard an airplane. On the other hand, **lithium batteries** are perfectly OK, 2-3 pieces on an average passenger nowadays. They are consumer-grade mass products without any serious certification. Any one of them (if it fails) is a potential **incendiary bomb**¹², **poison gas-smoke bomb**, and even an **explosive bomb** if you are very unlucky. Imagine the mayhem that it can cause inside an airliner’s cabin (and in a luggage hold as well). Try searching YouTube for “lithium battery explosion” and enjoy

¹² Unlike “**flammable**”, which denotes a high tendency to accidentally catch fire, the adjective “**incendiary**” actually refers to weapons deliberately designed to cause fire like napalm or thermite bombs or Molotov cocktails.

watching. Is the hazard of a Li-ion battery catching fire so low that it can be considered “highly improbable” or even impossible?

4. The danger of TEMPEST eavesdropping (more in detail later) is regularly neglected, even in high-security buildings. A usual excuse is that it requires too high-tech which is expensive. The first accidental TEMPEST eavesdropping dates back to 1916 - the equipment used was very low-tech. Most deliberate TEMPEST attacks have been mounted with less than \$20.000 of equipment.
5. Try watching the “Doomsday Preppers” series. Almost all disastrous scenarios are at least theoretically possible. At least 50% are correctly worked out with good quality solutions proposed, and from at least 25% you can pick up nice ideas from an engineering standpoint. Yet, most people will consider it to be “an impossible” conspiracy theory nonsense, even those with a real-life experience of **full-scale war**.

Electronic security, if mixed with espionage in any way is the last place where a “highly improbable” scenario can be neglected by Alice and Bob. Unlike natural forces, their enemies Eve, Mallory, and Walter must be considered unpredictable, devious, cunning, extremely patient, and well-funded, and expected not to play by any rules or laws whatsoever.

1.2.5 • The problems that electronic engineers can't solve

Cryptography solves only one part. Good electronic hardware design solves another. The same goes for carefully elaborated security procedures regarding the use of electronic equipment. Also for good technical education of your personnel. Electronic engineers can only build better electronic devices – everyone has to educate himself to attain a security-conscious mindset.

Apart from this, there are many other aspects of security that have nothing to do with electronic engineering or mathematics. Good electronic devices are necessary, but security is not all about high-tech.

For security to work, the end-user must be aware of threats- most of an average company's employees (even of one directly involved in security business) are either unaware or not motivated enough. This leads to negligence - the best crypto-electronics can't help here.

Read any of the books on Kevin Mitnick. Most of his exploits were about **social engineering**, the rest about programming and electronics. The social engineers' main rule is “If you don't have information, it's because you haven't asked for it yet.” An amazing amount of secret information can be extracted by simply **asking for it**. This circumvents any electronic security and cryptography - a target under deception usually has all the passwords and security clearance required. Electronic engineers can't help here. Sorry.

Kim Philby spent more than 30 years in the top ranks of the British intelligence spying for the Soviet Union. He was a spy who inflicted more damage to the west than any other. No cryptography or electronic engineering could have been of any help here. He had access

to all codes and passwords. British electronic engineers and cryptographers were highly skilled, of course. It wasn't about them, but other professions, like e.g. psychologists, to make a careful screening of job candidates to detect those suspicious.

Detecting possible spies is very difficult because people of very different psychological profiles start spying for very different reasons. Some spies did it simply for money- some of them were drug addicts who had to support their habit, others had a too demanding mistress. Others were social outcasts in the west who falsely believed that communists would always be on a side of an underdog. Some people feel that they don't belong to **any nation** - not directly associated with the usual national divisions of this world, they will cooperate with any side that seems OK. Some people simply believe in a certain ideology (a belief which is usually a self-generated justification) but based on different past experiences. Some spies ended WWII on the Axis side and survived - when they became involved in the Cold war, they did it mainly because they wanted to get a taste of victory (some on WP, some on NATO side!) - having lost one war was too much frustration they couldn't handle. Some did it because they found it exciting and high-tech. The list of unusual psychological profiles goes on and on.

This might be interesting, but as you can see, this is a job for someone else. In this book, I will concentrate on the problems that we **electronic engineers can solve**.

1.3 • Low tech rules - very unintuitive

Having read up to now, this should have become a little less unintuitive. In 1.1.3 we analysed how a higher level of integration of an IC decreases security. To prove the point, or to expand it a bit, let's make a security comparison of various long-term memory storage technologies throughout history. The main comparisons will be concerning:

- density of data storage
- price
- long-term storage reliability
- the possibility of fast and practical irrecoverable destruction of data (irreversible erasure or so-called zeroization)
- the possible existence of uncontrolled secret storage space
- susceptibility to TEMPEST eavesdropping
- possibility of detecting the author based on residual data

1. **Pen and paper** - cheap technology with the lowest density of data storage. If stored properly, will keep information for more than 1000 years - we've had enough time since the beginning of civilization to test it in every way. **Longer than any other technology.** Information on paper can be very easily zeroized by fire, ingestion, micro-shredders, or strong acid. There are no secret sectors on a sheet of paper, although there are technologies (like micro-dots and invisible ink) to hide information on paper -this will burn along with all other data when lit. Without any energy radiated, it can't be attacked by any means of TEMPEST. The only downside is that an author can be easily detected based on handwriting (learning to write with your other hand helps here). Altogether, **very reliable technology** from a security standpoint!
2. **Mechanical typewriter** - with a much better speed performance than pen and paper and a slightly higher price, it keeps most of the positive traits of its predecessor. It is also much harder to trace the text back to its author (analysis of handwriting is much easier), but still possible - the STASI were very good at this. Not only that they could link a typewritten paper to a particular typewriter, but also determine **the language** in which most of the plaintext was written. If the typewriter was used to write **ciphertext instead of plaintext**, it was even easier to detect, putting Alice immediately in a lot more trouble! How? Try analysing it yourself. If you are not sure, try again after reading the whole book: it will give you many leads!
3. **Electromechanical typewriter** - worse than 2.) - also transmits residual electrical pulses for another type of RF TEMPEST, and adds a possibility of planting an electronic key-logger as well.
4. **Analogue film tape** - stores more information per mm² than paper or magnetic tape, for a higher price. Requires chemical processing and can't directly convert information to electrical signals (like magnetic tape). Holds data reliably for more than 100 years. Burning or dissolving in a strong solvent will quickly destroy the data. **Microfilm** form (a film with higher storage density) has been any savvy spy's standard tool of the trade for many years. There is no possibility of secret storage space. An analogue camera has no useful TEMPEST radiation. On top of all this, it is very difficult to trace a film back to a camera! **A very reliable technology indeed.**
5. **Audio and video magnetic tapes** - both have been successfully used in past to store digital data. Reliable for more than 50 years, but prone to wear and tear, especially on low-quality tape decks. Special variants have been made to directly record digital data, required specially designed tapes and drives, for a much higher price. Fire or strong solvents quickly and effectively zeroize the data. Video tape systems with bandwidths up to 6MHz store data with much higher density than audio tape systems (up to 20kHz), but as a consequence, they radiate much more RF TEMPEST. There is no secret data space on audio tapes, but the latest generation VHS VCRs had some

metadata¹³ recording features. Because of this metadata, a certain VCR could leave its “fingerprint” to be detected later, which is almost impossible with audio cassette tapes. As you can see, **audio compact cassettes** are a more favoured tool: cheap and still easily available in the 21st century. We will address them later.

6. **CD and DVDs** - considered to be very suitable for long-term data storage at the beginning, but is now apparent they degrade after 25-30 years. Although their data density is much higher than tapes and the price may be even lower, they are not very favoured from a security standpoint. First of all, they are difficult to destroy. Hidden sectors are difficult to mount, but secret metadata (to identify the drive and author) can easily be hidden because the user loses direct control over data recorded to CD/DVD. TEMPEST radiation, especially when burning a disc, is high. Existence of read-only and read/write drives gives a possibility of ensuring data integrity after writing - can't be tampered with by use of a Trojan when used inside a read-only drive.
7. **UV EPROM** - easily available even nowadays, cheap, with an average data density, and very good from the security standpoint. Can reliably hold data for up to 30 years. Illumination with strong UV light destroys the data, but it takes up to 10 minutes. No secret sectors, maybe on some newest type, but not on standard ubiquitous 27C type that is still straightforwardly available. Metadata for identification is difficult to hide because users can easily control every memory block written. TEMPEST radiation is less than CD/DVD. Tampering with data in runtime by a Trojan is not possible, because 12-14 volts must be connected to a Vpp pin to write digital zeros, and UV light is required to revert all bits to digital ones.
8. **HDD discs** can reliably hold the data for up to 20 years, but it is recommendable to make earlier backups. Data density is very high, price is satisfactory. With the old types in CHS¹⁴ mode the user could have had some control of writing individual physical sectors, but this is fully lost with the new LBA¹⁵ mode. This means that undetected hiding of secret sectors and metadata is very easy, and difficult to detect. Secure zeroization is difficult, the only method to ensure irreversible deletion is heating up to Curie temperature, which is very impractical. Above all this, TEMPEST radiation is high due to the high frequencies involved. **A bad technology from a security standpoint.**
9. **SD cards, SSD, FLASH**, anything based on solid-state. The situation is bad enough with HDD, and here is much worse, in almost all aspects. They can't hold the data

13 metadata: additional data created to provide information about the main data stored on a medium. In case of a VCR, the main data is audio and video signal (or digital data encoded like PAL video, if VHS tape is used to backup digital data). Metadata could be a unique serial number of the VCR and time and date of recording, inserted by the VCR itself. Metadata inserted without Alice's knowledge can be very dangerous, especially if created by Mallory's Trojan planted to VCR's firmware.

14 CHS: Cylinder-Head-Sector - an old method of accessing data on a HDD by directly addressing its physical location on a disc.

15 LBA: Logic Block Addressing - a new method, addressing logic blocks, not physical locations directly. They can be routed to different physical locations on a disc.

reliably for more than 10 years. Price and data density are practically the best, but every other property is worse. Because individual memory cells tend to fail over time (an inherent property of SSD devices), the internal MCU keeps rerouting the data to alternative physical locations. The user here absolutely loses control of which physical sector of the SSD device is written and erased- the internal MCU takes care of this dynamically. A 16GB SD card is likely to contain more than 32GB of raw memory space - the rest is a reserve to gradually fill in the bad sectors as they fail. Some sectors are copied to several physical locations, all without any control from the user -this process is called **write amplification**. A secure zeroisation is possible, but only by heating to at least 1300°C, which is much higher than any material's Curie temperature and hence even more impractical. The presence of ABUNDANT storage space is necessary for SSD technology to OPERATE, for the reason of the inherent physics behind all low-voltage SSD devices. Abundant storage space is not needed for HDD, UV EPROM, cassette tape, and any other memory technology, but it is needed for SSD devices (like flash memory, SD cards, SSD drives...)

Secretly recording metadata for identification is also simple. Secretly changing critical data in runtime is very easy - unlike with 27C UV EPROM - FLASH and EEPROM can be erased and written with low voltage. **All in all, Alice's and Bob's nightmare, but Eve's and Mallory's sweetest dreams.** Regardless of all this, an SD card can still be used in secure applications, but only by observing certain limitations strictly. I will address this later as well.

Conclusion: Despite what James Bond movies say about high tech, it is low tech that rules when it comes to security.

1.4 • My design philosophy and approach to security

Now I will simply summarise the first chapter. These are the rules that Alice and Bob, our DIY spies on a low budget should follow to get good results:

- **use low-tech** whenever possible: it is still easily available, Eve can't monitor it, Mallory can't subvert it against you. Low-tech has managed to beat high-tech many times throughout history already. Even the old trustworthy Zilog Z80 will come in later to fill some security gaps, you'll see.
- **use a lower speed** of data transmission whenever possible - crucial messages are almost always short. Although Gb/s are available today, kb/s are often enough. There is less residual radiation for Eve to pick at a lower frequency because longer wires are required to effectively transmit RF energy at longer wavelengths. Filter square waves to basic harmonic sinewaves for the same reason if possible.
- **use a DIY approach**, don't always buy readily-made, off-the-shelf products
- **keep everything as simple as possible** - complexity is the main enemy of security
- **use easily obtainable, general-purpose components** whenever possible. Alice and Bob need to be able to get them everywhere, and without drawing much attention (purchasing special-purpose secure ICs will raise a flag for Eve).
- **use an open-source, open-hardware** approach, with a low level of integration, so every signal can be checked. This way you can test the security, which is necessary.

- **try to physically separate electronic modules** for different phases of crypto operations. This increases security, in the same way as low-level integration, enabling Alice to check and monitor more variables.
- **carefully read books and train yourself**, also on subjects other than electronics and cryptography - they alone can't solve your security problems.
- this will expand your scope of awareness and help you to **identify the real problems**
- never disregard any **"impossible scenario"**
- try identifying the problems that I and **other engineers may have overlooked**, and try to work them out.
- learn the basics of **microwave electronics**, although this may seem complicated (especially the mathematics behind it) - you need to understand the basics of the RF world to build proper defences.
- always remember that your security is primarily **your responsibility**
- build a security-conscious mindset - get into a head of a spy, learn to think like a spy. This means becoming a **professional paranoid**, while still keeping excessive paranoia away from your everyday life. Doesn't seem easy, and indeed it isn't.

This concludes the first chapter. The basic DIY principles of electronic security have now been outlined. In the next chapter, we will address specific problems and methods of attack that Eve, Mallory, or Walter can mount. We need to properly recognise the problems before solving them. We will also introduce Trudy, the intruder...

Emma: - Eileen, you are late! Your boyfriend is here already!

Eileen: - My boyfriend? What have you...

*Emma: - You were right, can't call him "Kestrel". His hands
are as gentle as a cat's paws...*

Who are these charming ladies? What are they talking about? Chapter 1 should have given you a lead to figure out. This was allegedly a true event. No worries, in case you give up, you'll find the full story inside this book. The same goes for some puzzle anecdotes that will come later.

Chapter 2 • Methods of Attack

Any crypto system can be attacked in many different ways. First, we have to see what common methods of attack can be mitigated using DIY methods, so we can then design a crypto-system and security procedures that simply can't be attacked by other methods that we can't mitigate.

2.1 • Methods to counteract

Let's start with the techniques that can be countered with a DIY approach:

1. **Mathematical crypto-analysis** - this means cracking encryption using mathematical/statistical calculations to narrow the search and find the key.
2. **Brute-force** - similar to previous, but less sophisticated - trying every possible key for decryption until a meaningful plaintext appears.
3. **Eavesdropping on/subverting/regenerating random numbers sequence** - every good encryption relies on good random numbers up to a certain extent. This method is about Eve trying to capture the sequence, or even regenerating it, or Mallory tampering with Alice's random number generator to create bad predictable sequences.
4. **Buffer-overflow** - if Alice's computer isn't carefully programmed/protected - i.e. input buffers for normal input data are not properly delimited in memory, then Mallory can input a longer sequence which can fill an intended part of the memory e.g. for stack or even program code. This way Mallory can steal Alice's data or even take control of Alice's computer.
5. **Side-channel attacks** - Eve can monitor Alice's device for power consumption, the acoustic or electrical residual signals it creates, measure its timing for different inputs, etc... After careful processing of the gathered data, Alice's critical secret data can be extracted.
6. **Software malware** (viruses, worms, Trojans, ...) - Mallory can plant all sorts of malicious software on Alice's computer, using different methods.
7. **Hardware Trojans** - the same as previous, but here Mallory manipulates the electronic components on a hardware level (e.g. like adding extra microcircuits on a CPU's silicon dies that will record/leak secret information) and then somehow plants them in Alice's lab/workshop.
8. **Exploiting inherently insecure physical properties** - this is similar to hardware trojans, but Mallory doesn't tamper with Alice's electronics - they are left physically untouched. If she and Eve capture some of Alice's electronics, will simply extract secret data, e.g. thanks to various unwanted data remanence effects, inherent to a certain type of electronic technology.

Now some of the attacks that we can't counter with a DIY approach are:

1. **Any type of side-channel or fault injection attack against secure hardware** - here the device under attack is e.g. some kind of a Smartcard (like a credit card), **with tamper-resistant features**, holding some kind of secret information (like a private key, or a secret program) that can't be normally accessed. Alice has lost the device and Eve has found it. Analysis of various residual variables (e.g. supply current - **power analysis**, or time needed for certain operations - **timing attack**) can lead Eve to reveal the secret data. This analysis usually requires highly complex mathematical processing of the measured variables. Sending invalid input data or electrical signals out of range (**fault injection**) to the device's input can cause undesired operation and may reveal secret data.
2. **All sort of invasive attacks (e.g. decapsulating¹)** - the same as the previous (Eve has got a hold of the Smartcard), but she will try decapsulating and inspecting it under a microscope, probing it with different electrical or optical signals to extract the secret data.

Mitigating these types of attacks requires specially designed secure integrated circuits, with different built-in countermeasures to mask the residual signals and critical memory areas, both physically and electrically. My crypto devices **will not rely on any specially designed tamper-resistant hardware holding any kind of secret**, simply because the development and production of tamper-resistant integrated circuits is very expensive and requires specialised equipment and workspace that Alice and Bob probably don't have. The encryption keys will be stored on general-purpose SD cards, paper, or cassette tapes and Alice and Bob will simply have to carefully guard them. Everything must be open-source (hardware and software), built with ubiquitous general-purpose components, all in line with my DIY design philosophy.

As you can now see, many of the threats can be countered on a DIY budget, while others can simply be made pointless, with certain well-defined design and security procedures. Now I will explain in more detail how to counter each one of the attack methods, all on DIY principles.

2.2 • Mathematical crypto-analysis

Unlike brute-force, this means using some mathematical/statistical methods to search through the key-space and quickly reject whole sets of incorrect keys - unlike brute-force where every possible key is tried.

For example, the **mono-alphabetic cipher** is a substitution of each letter of the plaintext with another one, always the same letter in a ciphertext. This means, for the English language (26 letters in the alphabet) we have $26! = 4.03e+26$ possible combinations which are more than 88 bits (because $4.03e+26 = 2^{88.38}$). This is too many combinations to brute-

¹ decapsulation: opening an IC, removing a plastic package, usually with nitric acid and acetone. The goal is to expose the silicon die without damaging it, so the microcircuits can be inspected with a microscope or probed with electrical and optical signals.

force, but there is no need since the fast crypto-analysis method for mono-alphabetic cipher has been known since the 14th century -it was described exactly by Arab cryptologist **Ibn al-Durayhim**. Now may be a good time to remark that the world's first book on crypto-analysis was written in the 9th century, also in the Arabic language. English word **cipher**, the same as in many other languages, also comes from Arabic - while the well-known word **algorithm**² comes from Persian.

Some letters have a different frequency of occurrence in a plaintext written in a particular language. Above this, some letters have different frequencies as the first or last letter in a word, and the same goes for particular 2 or 3-letter groups (digraphs or trigraphs).

Order Of Frequency Of Single Letters	E 12.7 T 9.1 A 8.2 O 7.5 I 7.0 N 6.8 S 6.3 H 6.1 R 6.0 D 4.3 L 4.0 U 2.8 %
Order Of Frequency Of Digraphs	th er on an re he in ed nd ha at en es of or nt ea ti to it st io le is ou ar as de rt ve
Order Of Frequency Of Trigraphs	the and tha ent ion tio for nde has nce edt tis oft sth men
Order Of Frequency Of Most Common Doubles	ss ee tt ff ll mm oo
Order Of Frequency Of Initial Letters	T O A W B C D S F M R H I Y E G L N P U J K
Order Of Frequency Of Final Letters	E S T D N R Y F L O G H A K M P U W
One-Letter Words	a, I
Most Frequent Two Letter Words	of, to, in, it, is, be, as, at, so, we, he, by, or, on, do, if, me, my, up, an, go, no, us, am
Most Frequent Three-Letter Words	the, and, for, are, but, not, you, all, any, can, had, her, was, one, our, out, day, get, has, him, his, how, man, new, now, old, see, two, way, who, boy, did, its, let, put, say, she, too
Most Frequent Four Letter Words	that, with, have, this, will, your, from, they, know, want, been, good, much, some, time

Table 2.1 - frequencies of the most common letter groups in the English language

This will quickly narrow the search, and is possible with the pen and paper method, without a computer or even a calculator (14th century low tech). There are much more detailed statistics calculated for every language - e.g. in English, the easiest to detect consonant is "N", because it is almost always preceded by a vowel (or located at a beginning of a word) - in 80% of words.

Even the most powerful cipher (with the largest key space, absolutely impossible to brute-force if used properly), the **Vernam's cipher**, also known as **one-time pad (OTP)** - not

² Abu Jaffar Muhammad bin Musa al-Khwarizmi, 9th century Persian mathematician. His last name was first translated to Arabic as al-Khwarizmi, then to Latin as Algorithmus.

to be confused with one-time passwords, which is something different) can be **crypto-analysed** if two different meaningful plaintext messages are encrypted with the same random key sequence.

This is how OTP works. Let P denote the plaintext, K the key-the random numbers sequence the same length as the plaintext, C -the ciphertext. Encryption is bitwise “exclusive or” between P and K ($C=P\oplus K$), and decryption is the same operation $P=C\oplus K$. Exclusive or is a completely **linear operation**, so the following is easy to calculate:

$$P = C\oplus K, \quad P = P\oplus K\oplus K = P, \quad K\oplus K = 0, \quad P\oplus 0 = P.$$

With a good random numbers sequence K , every ciphertext can decrypt to any plaintext, and therefore there is no possibility of crypto-analysis or brute-force attack. If, however, Alice encrypts two different meaningful messages, $P1$ and $P2$ with the same sequence K , Eve gets a chance to crack the code after intercepting ciphertexts $C1$ and $C2$:

$$C1 = P1\oplus K, \quad C2 = P2\oplus K, \quad P2\oplus C2 = P2\oplus P2\oplus K = 0\oplus K = K, \\ P1 = C1\oplus K = C1\oplus P2\oplus C2 = C1\oplus C2\oplus P2 = C\oplus P2.$$

Eve first calculates $C=C1\oplus C2$ and then tries different meaningful words (helped by more detailed versions of table 2.1) and sentences for $P2$. A meaningful $P1=C\oplus P2$ will start to emerge, and Eve will advance with subsequent $P2$ words yielding meaningful $P1$ words, until the end of the shorter message ($C1$ or $C2$).

New and better encryption methods are constantly being invented, but as mathematics and computers are also advancing, new methods of crypto-analysis are catching up. For example, the public-key RSA method is difficult to crack, because multiplying two large prime numbers is easy, but factoring a big **semi-prime** (a composite number with only two prime factors) is much more difficult. As new mathematical methods (like e.g. elliptical functions or Shor’s algorithm) and computers (like **quantum computers**) are advancing, this is becoming easier. Sometimes a new mathematical crypto method is invented; it seems nice and strong but after a few years several weaknesses are found which enable Eve to crack the carefully saved old messages without much brute force.

2.2.1 • Brute-force

The name of the method is self-explanatory. Simply try all the possible keys, until you get a meaningful plaintext. For example, Caesar’s cipher is easy to brute-force, because there are only 25 possible keys for the 26-letter alphabet (Caesar’s cipher is a simple special case of the general mono-alphabetic cipher, a simple shift by adding a fixed number - Gaius Julius Caesar used +3).

If the ciphertext is “M SAAP EBK FDGEFE ZA AZQ MZP DQYMUZE GZWZAIZ”, it is very easy to try all the 25 possible keys:

+1	L RZZO DAJ ECFDED YZ ZYP LYO CPXLTYD FVYZHY
+2	K QYYN CZI DBECDC XY YXO KXN BOWKSXC EXUXYGX
+3	J PXXM BYH CADBCB WX XWN JWM ANVJRGB DWTWTFW
+4	I OWWL AXG BZCABA VW WVM IVL ZMUIQVA CVSVWEV
+5	H NVVK ZWF AYBZAZ UV VUL HUK YLTHPUZ BURUVDU
+6	G MUUJ YVE ZXAYZY TU UTK GTJ XKSGOTY ATQTUCT
+7	F LTTI XUD YWZXYX ST TSJ FSI WJRFNSX ZSPSTBS
+8	E KSSH WTC XWYWXW RS SRI ERH VIQEMRW YRORSAR
+9	D JRRG VSB WUXVWV QR RQH DQG UHPDLQV XQNQRZQ
+10	C IQQF URA VTWUVU PQ QPG CPF TGOCKPU WPMPQYP
+11	B HPPE TQZ USVTUT OP POF BOE SFNBLOT VOLOPXO
+12	A GOOD SPY TRUSTS NO ONE AND REMAINS UNKNOWN
+13	Z FNNC ROX SQTRSR MN NMD ZMC QDLZHMZ TMJMNVM
+14	Y EMMB QNW RPSQRQ LM MLC YLB PCKYGLQ SLILMUL
+15	X DLLA PMV QORPQP KL LKB XKA OBJXFKP RKHKLTK
+16	W CKKZ OLU PNQOPO JK KJA WJZ NAIWEJO QJGJKSJ
+17	V BJYJ NKT OMPNON IJ JIZ VIY MZHVLDN PIFIJRI
+18	U AIIX MJS NLOMNM HI IHY UHX LYGUCHM OHEHIQH
+19	T ZHHW LIR MKNLML GH HGX TGW KXFTBGL NGDGHPG
+20	S YGGV KHQ LJMCLK FG GFW SFV JWESAFK MFCFGOF
+21	R XFFU JGP KILKJ EF FEV REU IVDRZEJ LEBEFNE
+22	Q WEET IFO JHKIJI DE EDU QDT HUCQYDI KDADEMD
+23	P VDDS HEN IGJHIH CD DCT PCS GTBPXCH JCZCDLC
+24	O UCCR GDM HFIGHG BC CBS OBR FSAOWBG IBYBCKB
+25	N TBBQ FCL GEHFGF AB BAR NAQ ERZNVAF HAXABJA

If the ciphertext is created with RSA and the public key is known, factorisation of the big public semi-prime number ($n=p*q$, p , and q are prime) can recover the private key. If done simply by trying to divide with every integer number from 2 till $n/2$, it is brute-force. Of course, it doesn't work, because it would require too much computing power.

This example of Caesar's cipher was a so-called **ciphertext-only attack**. The cipher was cracked knowing only the ciphertext. Modern crypto algorithms are too strong to be broken like this. The **known-plaintext attack** is more powerful - here Eve knows a part of the plaintext or some plaintext of Alice's previous messages and uses it to speed up the search for Alice's key. Known plaintext may be a header of some particular type of file - e.g. all the ***.docx** or ***.bmp** files start with known common bytes.

This is why I decided to use OTP in my systems: its mathematical background is extremely simple and is the only method that can be **mathematically proven as unbreakable**,

as long as it is properly used, with a good RNG for generating keys - I will address this problem later. It can't then be crypto-analysed or brute-forced, not even in theory. **Known plaintext is useless to crack it**, even with the most advanced quantum computers.

2.2.2 • Attacks on RNGs

As I already said, RNGs are extremely important for a good crypto system. Random numbers are used in almost every modern crypto system, in different forms and functions. If the crypto-algorithm is OTP, we need a very long sequence of random numbers (one-time pad) as an encryption key. A PGP program uses random numbers to determine prime numbers to generate public and private keys. A random number generated by a network client and encrypted with a server's public key then sent encrypted to the server can be used in some symmetric encryption algorithm (e.g. AES) as a one-time session key.

There are different types of RNGs, but **very few** of them are suitable for use in cryptography. This will be explained in detail in chapter 3, but for now, we will see how Eve and Mallory could try to exploit possible weaknesses in Alice's and Bob's RNGs. We can also assume the generated random sequences are kept secret, but the technical details of Alice's RNG will eventually become known. The same usually happens with crypto algorithms- the mathematical details of, for example, AES or RSA are publicly known; only the keys must be kept secret.

Here are some possible methods. You will quickly see that Eve and Mallory have even more chance of success if Alice and Bob work for a big three-letter agency than alone on a DIY budget:

1. **Eavesdropping on RNG** - possible if running on a general-purpose PC or networked device. Stealing data from a networked device is much easier than from a stand-alone offline device - and yet many devices are unnecessarily networked and nowadays online 24/7 (e.g. **online bitcoin wallets** - they "securely" store your private keys on servers accessible 24/7 online). The RF TEMPEST method is also possible if the type and physical location of the RNG is known to Eve.
2. **Re-generating the sequence** - if PRNG was used. Good true random sequences can't be re-generated by any method. Pseudo-random numbers are calculated from previous values - this is how my Speccy did it - any digital computer is too deterministic to produce truly random numbers without extra circuitry, which was too expensive in the 1980s. Although PRNGs are bad for crypto applications, there were crypto devices produced in the past which used PRNGs regularly re-seeded by some slower (allegedly **true random**) process.
3. **Tampering with RNG itself**- i.e. rigging it by Mallory - to transmit sequences to Eve, or to generate weak sequences, or planting pre-generated sequences. This can be done by rigging the RNG device before delivery to Alice and Bob - or more easily by rigging the components being delivered to Alice's workshop.

4. **Jamming the RNG** with signals deliberately transmitted by Mallory. If Mallory is close enough and knows that Alice's RNG is powered from AC mains with a badly filtered power supply, she can try sending jamming signals through the AC mains. Transmitting RF jamming signals with a directional antenna can also work if the RF-EMI³ shielding of the RNG is poor.
5. **Tampering with the delivery of one-time pads** to Alice and Bob - Mallory plants her own - only one pad for Alice and one for Bob are needed. This is way easier to pull off within big agencies, where a third person always generates keys for Alice and Bob.

2.3 • Buffer-overflow

This type of attack is all about a badly coded program on Alice's computer, which doesn't properly limit the input buffers. It seems very easy to prevent: all we need is one or two lines of code for counting input bytes and stopping when the input buffer is full. Buffer-overflow may be viewed as a special type of **fault injection attack** which means **inputting badly formatted data** (correct voltage, pulse lengths, frequencies, and bits in a frame, but simply too long a sequence of bytes).

We will see that it may seem easy, but it isn't - especially on today's computers, considering the way they are programmed and operated.

1. Let's start with a **simple computer**, on the level of typical 1970s or 80s technology. One CPU (e.g. Z80 or 6502), several KB of RAM, several KB of ROM (or UV EPROM), and a few simple I/O units. **No operating system**. Only a few interrupt service routines to scan a keyboard and to drive a monochrome VDU⁴. ROM and RAM mappings are easy to check - this way you know exactly which parts of RAM you can use for your program, and which you can read, but never write to. You know which part of RAM is used for stack - then you know how many consecutive subroutine calls you can make before **overflowing the stack**. You know which part is video memory. You know where the keyboard scanning routine is located. Cassette tapes are used for storage and can be manually operated - **no need for an operating system**. Consequently, if you code your cryptography program in **assembler**, it is relatively easy to delimit the RAM areas and avoid buffer overflows.
2. Now let's try the same as 1. - but with a **high-level language (HLL)**. We won't use BASIC, Pascal, or Python for secure crypto. We will use C. Programming in C enables us to write simple lines of legible code, while still maintaining good control over generated machine code and memory areas used. This way it is still relatively easy to prevent buffer overflows.
3. As disk drives became more readily available, it was decided that computers needed operating systems. They were used primarily to coordinate low-level disk operations. It is also easier for programmers to write programs in high-level languages intended

3 EMI: electromagnetic interference

4 VDU: a visual display unit - an old name for a computer monitor

to run on top of an **operating system** taking care of low-level jobs. On the other side, security pitfalls started to emerge because now programmers started to lose control over raw memory space (OS took over this task, among other low-level jobs) - it became more difficult to mitigate buffer overflows.

4. It was then decided that **multiple applications** needed to be run simultaneously on one computer, under one operating system. This usually means dynamically allocating RAM for each application -now it was even harder to control buffer limits.
5. Next, there came multiple **applets** installed on top of each application. Then it became common to frequently install **plugins** for each applet. All of this may require frequent updates, all downloaded and installed from the internet. Now it is practically impossible to reliably control input buffer limits.

Almost all the general-purpose PCs have been running in mode 5.) for the past 20 years. It is easy to see that the best defence against buffer-overflow attacks for Alice and Bob is to use very simple systems, without an OS, in mode 1 or 2. They can still use networked PCs or smartphones for less critical tasks while letting specially designed simple crypto devices handle all the security-critical operations. Starting from chapter 3, I will explain which devices are needed and how to build them. First, we need to complete identifying all the possible threats for Alice and Bob (until the end of this chapter), before starting to solve the problems and design specific crypto-devices.

To summarise this section, for mitigating buffer overflows, the best strategy is to use simple systems, without any OS, with one single application process running, programmed directly in assembler or C, to carefully maintain control over every byte of memory. Needless to say, this will also protect Alice and Bob against any kind of **software malware** (viruses, worms, Trojans...) which will then be very difficult to plant.

2.3.1 • Types of buffer-overflow attacks

Executing a successful buffer-overflow attack requires good knowledge of the target computer's machine code, assembler, and hardware. The same goes for defending against it. You can be a good C programmer (to create good functional programs in C) without ever learning your CPU's assembly code and architecture, however, you won't be able to make it secure against Mallory's attacks.

Overflowing the input buffer can harm the target computer in many different ways. Let's go through a few variants.

1. **Smashing the stack:** the simplest variant is to simply fill input buffer with a long sequence of zeros - the zeros will fill the stack area where the subroutine return addresses are normally stored. This will make the program jump to 0x00000000 memory address when it completes the subroutine (reaches RET command), which is usually equal to reset/restart. Just resetting the computer in the middle of a critical operation may be enough to cause serious harm to Alice. If the memory address map

of Alice's computer is known to Mallory, she can opt for a more subtle variant like planting a fake return address to the stack which will start a program that Mallory can better exploit.

2. **Code injection:** instead of just planting return addresses on the stack, Mallory will also inject arbitrary code to be executed by Alice's computer, which will enable Mallory to steal some crucial data or even take control over Alice's computer. The injected code is a carefully assembled machine code, to be run on Alice's CPU. Good knowledge of assembler and particular machine code is a must. Mallory will inject the **payload** (the arbitrary program code indented to execute some task for Mallory) and (usually after a sequence of useless NOPs) the fake return address to the stack, which will make a jump to the beginning of her injected arbitrary code. There are different variants of this attack, depending on the target computer's architecture and memory map. Arbitrary code can be injected into the RAM area normally reserved for the stack, but if the computer has protection against executing the code from the stack (normally used just to store temporary variables and return addresses), this won't work. Mallory doesn't know the exact positions of Alice's stack and input buffers in the memory nor the exact value in the stack pointer, so she must make educated guesses and hunt in the dark to successfully plant her injected code.

2.3.2 • Von Neumann's vs. Harvard architecture

These are two very basic types of computer architectures. Let's make a comparison, with emphasis on susceptibility to buffer-overflow attacks.

A von Neumann's computer stores instructions and data in the same memory area. Its CPU needs to control only one data bus and one address bus for both. Since there is only one data bus, it needs more clock cycles to fetch instructions and data. The memory allocation is more flexible: a programmer can easily change the balance between data and instruction memory. A typical CPU in this architecture is e.g. Zilog Z80.

A Harvard computer stores instructions and data in different memory areas. Its data bus controller is more complex because it needs to coordinate two data busses: one for instructions, one for data. This means that an instruction and data can be fetched simultaneously. Data and instruction memory areas are more rigidly delimited by hardware. MCUs (like e.g. Atmel AVR) usually have this architecture.

To be more precise, AVRs belong to a so-called "**modified Harvard architecture**". Flash memory stores the program, while SRAM is a volatile data memory. However, it is still possible to use **lpm** (load program memory) instruction to read data from flash (usually used to store look-up tables for calculations or lines of text to display on an LCD) - this means that instruction memory can be read like data memory, but slower. On newer models, the **spm** (store program memory) instruction was introduced, which can write to flash memory in run-time, with no need for high voltage (12-14V, like for UV EPROM). This is a slow process, and also a low endurance of flash memory (1000-10000 write cycles) must be taken into account.

Harvard computers (even those “modified”) are less susceptible to buffer-overflow attacks than von Neumann’s. It is still possible to smash the stack and change crucial data in SRAM, but it is not possible to inject the arbitrary code into instruction memory and execute it. It would require the very slow execution of *spm*-like commands. Alice can notice it interfering with normal CPU operation. It should be pre-loaded to flash to work, which is a chicken and egg like problem. This is one of the reasons I started designing my secure devices on Atmel AVRs. Later, as I dove down into deeper depths of professional paranoia, I decided that it isn’t secure enough, mainly because of the too high level of integration (CPU, flash, SRAM, EEPROM, WDT, IO, ADC, CTC, etc. all on the same IC) and the possibility of being re-programmed with low voltage in run-time by that notorious **spm** instruction. Then I considered the Z80 (old and reliable, easier to program in assembler because of **CISC**⁵ concept, produced in millions, well known silicon die blueprints, difficult to trojanise, lower level of integration,) with UV EPROM (impossible to re-program in run-time without high voltage and/or UV lamp), but I had to find a way to mitigate security pitfalls of von Neumann’s architecture regarding buffer-overflow attacks. You will see later in chapter 5 that there is a simple hardware solution for this problem.

2.4 • Side-channel attacks

Besides inputs and outputs that devices are designed to process, there are always other **residual signals**, or more generally **variables** that can be measured on a given device, although it wasn’t designed to emit these signals. Here are some examples:

1. **Mechanical typewriters** are designed to print letters on paper, not to create loud acoustic signals (it took a lot of work to silence them. This was the main source of stress and frustration for office workers until the advent of more silent printing technologies). The acoustic noise of a typewriter is its **side-channel**. Typebars have slightly different lengths, and different mass in their heads - depending on a letter (it is easy to understand that a letter “**M**” typebar has a little heavier head than the letter “**I**”) - so the frequency spectrum of the sonic “bangs” they make is different. Careful Fourier analysis of acoustic signals may reveal the letters typed.
2. **Credit cards** and other Smartcards sometimes need to internally (off-line) process PINs when entered from a terminal for verification. Perhaps processing of a correct decimal digit will take a little longer or shorter than the processing of the other 9 incorrect digits - this is a so-called “**timing attack**”, meaning a 6-digit PIN can be cracked in only 60 attempts (compared to 1 million needed for a **brute-force attack**). If a power supply current to the Smartcard is measured and compared when processing different decimal digits - maybe a little higher or lower current is needed to process a correct digit, this is a “**power analysis attack**”. Furthermore, if each incorrect

5 **CISC**: complex instruction set computer, as opposed to **RISC** which stands for reduced instruction set computer. CISC has many more assembly instructions implemented than RISC, thanks to more complex hardware (instruction decoder and execution circuitry). They are easier to program directly in assembler, because of a more versatile set of instructions. Zilog Z80 is CISC. RISCs are designed primarily for programming in **high-level languages (HLL)**, since it is easier to efficiently compile a HLL code to a reduced instructions set. Atmel AVRs are RISC.

attempt initiates a write to internal EEPROM (to decrease the PIN try counter), Mallory can cut the power to the card (a write to EEPROM requires a few more mA for a few ms) immediately, and then try again. Mitigating these attacks is difficult and requires specially designed hardware and very careful assembler coding to make the program always execute at the same time and spend the same amount of energy. The timing of various assembler instructions for some CPUs is easy to calculate, then insert a few NOPs to equalize the execution time... but then the power consumption is not the same! NOP and ADD machine instructions don't need the same amount of energy to fetch, decode and execute (ADD needs more energy). If the Z flag is set, it will take a little more energy (to flip that flag register bit - "Z") than if it remains zero. The usual strategy is if it takes 100µs to check the PIN, extend the operation to 10ms (100x more time) - to mask the supply current signal with 99% of useless obfuscated code and also add some pseudo-random noises. As I said before Alice and Bob will simply avoid using this type of secure hardware since they don't have the expensive resources required to build adequate defences.

3. **Old CRT⁶ video screens** used a sweeping electron beam to draw their picture, sequentially line-by-line. At PAL standard it was 25 frames per second, and each frame consisted of 2 fields (first odd then even lines drawn) to make the flicker at 50Hz more bearable to human eyes. Because of the relatively high amounts of energy required, there were a lot of residual electrical and optical signals emitted, that Eve can analyze, to "read" Alice's plaintext from her screen.
4. **Cables carrying electrical signals:** Analogue and digital, always transmit some RF energy. RF radiation becomes effective when the length of cables reaches at least 1/4 wavelength, but this is a common situation for many standard types of signal. If there are some unused wires in the cable, and if internal shielding is bad, Eve can pick Alice's transmissions simply by amplifying and filtering the **crosstalk⁷**.
5. **A telegrapher's "signature"**, as analysed in 1.2.2, may also be considered as a type of a side-channel - a unique rhythm of keying leaks a lot more information (about both operator and type of Morse key used and even RF gear used. It also affects the modulated **ring-out⁸**) than just pure dots and dashes that are intended to be transmitted.

Besides side channels, **subliminal channels** are also worth mentioning. Side channels are related to the **accidental** leakage of information, because of the inevitable physics of devices. Subliminal channels can be viewed as **side channels built on purpose**. Mallory can plant additional malicious code inside Alice's crypto software to inconspicuously leak her private keys, e.g. add an extra pause between the bytes transmitted to leak a digital one and no extra pause to leak a digital zero. After several hundred bytes of message,

6 CRT: cathode ray tube

7 Crosstalk: parasitic capacitance and mutual inductance between badly shielded wires closely spaced, will enable to induce unwanted voltage and current signals among wires

8 ring-out: residual oscillations, mostly unwanted, usually when abruptly switching on/off - e.g. with a Morse key. Can be mitigated with proper filtering.

Alice's private key will also be transmitted undetected by Alice and Bob. Eve will just have to listen in and arrange the bits together. Subliminal channels may also be viewed as a special type of well-concealed planted Trojans. The defence against **subliminal channels** is practically the same as that against **buffer overflows** and **malicious software**. If Alice and Bob build and program a simple system themselves (as described in 2.3), Mallory won't be able to plant a subliminal channel.

2.4.1 • TEMPEST - a type of side-channel

The word "tempest" means a violent storm. It was also the name of several American and British warships and warplanes. Some say that it is an acronym for "**Telecommunications Electronics Materials Protection from Emanating Spurious Transmissions**", while others disagree mainly because residual emanating signals are not necessarily electronic, i.e. purely **acoustic TEMPEST** attacks (like e.g. listening to a mechanical typewriter) are also possible. You may prefer other variants like "**Transmitted Electro Magnetic Pulse / Energy Standards and Testing**" or even "**Tiny Electro Magnetic Particles Emitting Secret Things**".

In any way, it is a subset of side-channel attacks. A timing attack on Bob's Smartcard in Mallory's possession is not considered a TEMPEST attack. Also, the identification of a telegraph operator is not a TEMPEST attack. The RF telegraph signal received is not residual, but meant to be transmitted. Remote eavesdropping on unwanted residually generated electrical, optical or acoustic signals is a TEMPEST attack. TEMPEST is mounted by Eve, not Mallory. It is about listening to accidental residual emanations and decoding them, not about injecting faults, stealing, or rigging Alice's hardware to leak data.

Let's go through a few historical examples in detail.

1. **WWI, 1916, the battle of Verdun** - one of the longest and the most costly battles in history. The trench warfare of WWI used very long static installations of trenches, tunnels, fortresses, and other buildings. Wired field telephones were widely used as a reliable means of communication. Kilometres of telephone wires were laid along trenches, usually buried in earth in very shallow depth, less than 1 meter. Shielding and twisting of pairs of wires, as means of reducing interference and crosstalk, were not enforced seriously at the time because electronic warfare was still in its infancy. Besides, sources of noise and interference were far from omnipresent like today. Audio-frequency vacuum tube amplifiers and filters were well developed in theory and practice, and available.

The stage was well set for the first accidental TEMPEST attack in history. The battle lasted for almost 10 months, most of the time with very small advances on both the French and German sides. After the French army managed to capture one German trench, they laid their field telephone cables. Unknown to the French, a German telephone cable was still there, buried in the close vicinity.

Let's check some basic math first. Regardless of frequency, there will always be some crosstalk between two closely spaced cables. Basic transmission line theory says that

it takes a line at least $\frac{1}{4}$ wavelength long, for the crosstalk to become significant. Light velocity in a vacuum is 300.000 km/s, but is reduced down to 200.000 km/s on many types of cables (since, $v = 1/\sqrt{LC}$ magnetic permeability remains close to vacuum, while dielectric constant is significantly higher). The frequency of human voice is in a span between 200Hz-3000Hz. At 2000Hz, the $\frac{1}{4}$ wavelength is 25km, which may not seem too long for a WWI trench, but a 200Hz component would require at least 250km.

How did it work? There is also another factor to consider: the cables were buried in plain earth and this normally contains a variety of mineral salts with significant conductance, especially when wet. High conductance of surrounding media **will increase the attenuation of a signal radiated** from one cable to another (the frequency is low, so optimistically for the Germans, it could have reached a 1-2 meter distance), but will also help to significantly decrease the wavelength.

Let's check the basics of microwave electronics now (within the aforementioned first part of the textbook [11] that will also give you sufficient knowledge to begin to debunk the stealth aircraft myths, as engineers from Serbia did very well in 1999). Let's take $\sigma=0.1\text{S/m}$ as an average conductivity of relatively dry earth, at a temperature above freezing point. Let's take a relative permittivity, $\epsilon_r=100$, which is in the worst case, much higher than seawater. If the condition $\sigma \gg 2\pi f \epsilon_0 \epsilon_r$ is met, we can consider the medium to be a good conductor. The highest frequency of interest is 3000Hz, and hence $0.1 \gg 1.67 \times 10^{-5}$, so we can consider earth a good conductor for the frequencies of interest with a very high margin - even when the earth is frozen!

In the case of a good conductor, the equation for a complex wave propagation factor

γ becomes simply $\gamma = \alpha + j\beta = (1+j)\sqrt{\frac{\omega\mu\sigma}{2}}$, where $\omega=2\pi f$, and μ is a permeability constant. Penetration distance δ (a distance travelled within a medium, where amplitude decreases to 37%) is calculated from the damping factor α as $\delta=1/\alpha$. Since $\beta=2\pi/\lambda$, the wavelength is then simply $\lambda=2\pi\delta$. The penetration distance for $f=3000\text{Hz}$ is then $\delta=29$ meters, and for $f=200\text{Hz}$ an even higher $\delta=112$ meters. This part works well. The attenuation for 3000Hz is less than 7% at two meters distance between cables.

Regarding the wavelengths: the kilometres in a free-hanging cable now decrease to only $\lambda=182\text{m}$ for 3000Hz and $\lambda=705\text{m}$ for 200Hz. This means that when buried inside plain earth, we can expect very good crosstalk (good for Germans, bad for French) over a few kilometres of trench (the theoretical $\frac{1}{4} \lambda$ minimum required is less than 200m) - definitely not very long for WWI trenches. By significantly reducing the wavelength, at the cost of insignificant attenuation, an otherwise very bad RF material (plain earth) works well to improve the situation!

Furthermore, it is worth noting that the speed of propagation between two cables ($v=\lambda f$) is now only 141 km/s for 200Hz and 546 km/s for 3000Hz component, far slower than the speed of light.

Very unintuitive, isn't it? Almost like shooting down an invisible stealth airplane with

a 1960s missile system... After noticing some strange weak noises on their telephone line, the Germans called for technical support, and connected an audio vacuum tube amplifier with an adjustable corrective filter to equalise the audio frequencies, for amplitude, but more important also for the different delays - since 200Hz component travels almost 4x slower than 3000Hz through earth. French conversations became audible enough, and the world's first electronic TEMPEST attack was achieved.

2. **Cold War, 1985, Netherlands** - Wim van Eck has just finished writing a report [9] about a successful TEMPEST attack against a common CRT computer monitor (VDU), at a range of several hundred meters. The attack method which was still considered only theoretically possible would soon make history, and be called "Van Eck phreaking".

A monochrome VDU displays only bright (binary 1) or dark (binary 0) pixels. In the PAL video system, one frame contains 625 horizontal lines, displayed as 2 fields (1 field contains odd line numbers, 1 field contains even), one field every 20ms. It takes 64μs to draw one line (15.625kHz-**horizontal frequency**). If it is possible to display 600 pixels per line, each pixel has a time window of 106ns (it is less because in 64μs only 52μs are available, while 12μs is used for HS-horizontal sync pulse). The abrupt luminance pulse (one for each bright pixel - bit 1) should not last longer than 50ns, because of the properties of a CRT screen. A frequency spectrum of a single square pulse (a so-called "box pulse") of duration $T_b=50\text{ns}$ can be calculated using the Fourier integral. The power $P(f)$ at the frequency f equals:

$$P(f) = A \left(\frac{\sin(\pi f T_b)}{\pi f T_b} \right)^2$$

As you can see in figure 2.1 (with logarithmic amplitude scale in decibels), an abrupt square pulse, designed for a PAL video system VDU, with a channel bandwidth of 7-8 MHz, will have significant radiation in the entire VHF band (up to 300MHz) and even (at only -40dB attenuation) in the low UHF band. **The envelope decreases more slowly at higher, UHF frequencies.** Each lobe is 20 MHz wide, with a -3dB bandwidth of more than 8MHz which is more than enough for transmission of a standard PAL TV video signal.

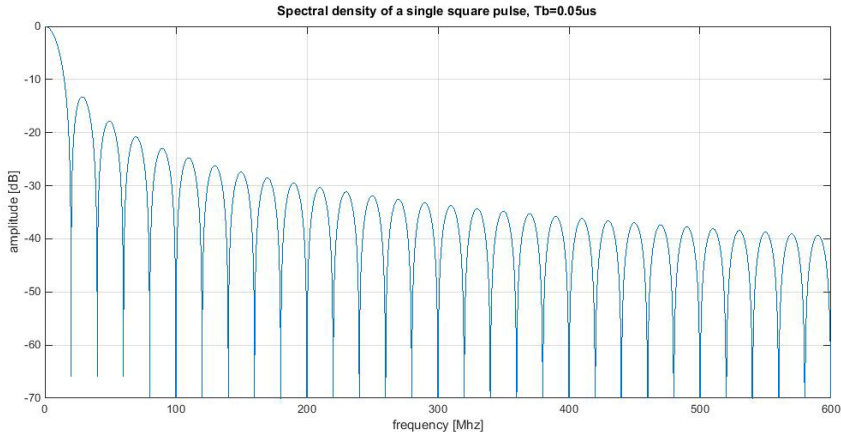


Figure 2.1 - Spectral density of a single square pulse of duration $T_s=50\text{ns}$

In the other words, a standard VDU, without any protective measures in place will transmit pixel-pulses all over VHF and low UHF range, at low RF power. Furthermore, because of VDU's particular physical conductor arrangement, **some lobes may resonate** and transmit higher RF power than other lobes. At 300 MHz, the $\frac{1}{4}$ wavelength is 25cm. and at UHF is even shorter. This is a length of many conductors likely to be found inside an average VDU.

It is important to mention that a standard PAL VDU is an electronic device designed to operate with input signals of a maximum 7-8MHz bandwidth (standard PAL signal). **Its purpose is not to transmit or modulate any RF power.** The internal pixel-pulse generator will generate 10MHz pulses (50ns pulse, 50ns space), and this is the highest frequency needed to enable the VDU operation. On the other hand, because of the inherent pitfalls of this kind of technology (abrupt high voltage pulses needed to modulate the luminance inside the CRT), the unwanted signals will be generated and transmitted at low power even up to a GHz range!

Conductors and metal parts, in **any electrical apparatus**, may behave as **parasitic helical resonators** (figure 2.2). Helical resonators are LC tank circuits-filters, with capacitance and inductance implemented in the same elements i.e a coil and box walls (a so-called **distributed** filters, as opposed to **lumped** type design where L and C elements are physically separated parts), typically designed for 100MHz-1GHz range, although frequencies as low as a few MHz are also feasible, but not very common. According to Zverev's formulas, a helical resonator designed for 100 MHz with -3dB bandwidth of only 1MHz will have the following dimensions: helix diameter $d=3.2\text{cm}$, helix length $b=4.8\text{cm}$, number of turns $N=8$, wire diameter 3mm, inside a shield-box with a diameter of $D=10\text{cm}$ and height of $H=8\text{cm}$. A parasitic configuration of conductors similar to this (with a much lower Q factor) is likely to exist inside a VDU enclosure, and even more likely for higher frequencies, for which smaller dimensions and fewer turns are required.

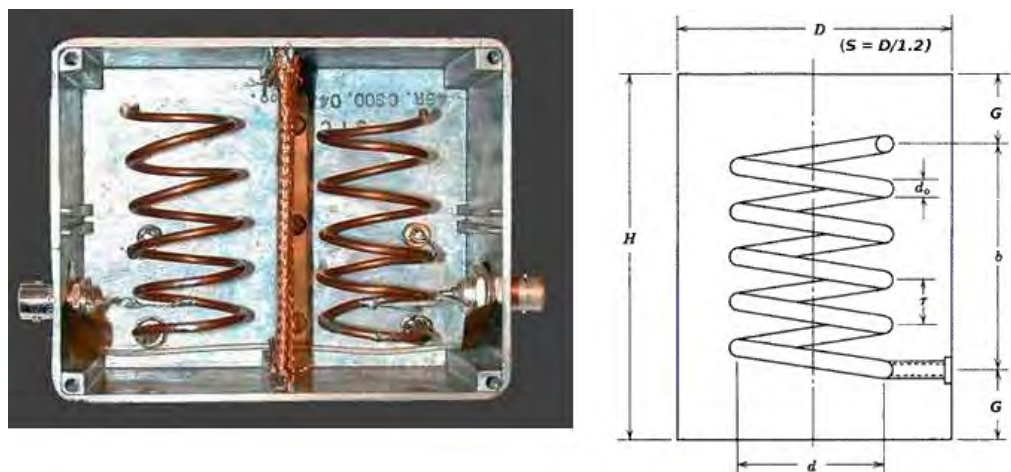


Figure 2.2. - A helical resonator, photo, and drawing

This is why it is possible to eavesdrop on two identical VDUs located inside the same room from 100 meters distance. They may have a slightly different internal arrangement of conductors - i.e. different resonant frequencies of parasitic helical resonators, **so they will “transmit” on different resonant RF frequencies.** Besides internal conductors, external cables (both signal and power supply cables) can also behave also as antennas and resonators, and resonate at different RF frequencies for the same reasons.

This is how RF pulses for each pixel are transmitted, all over the VHF and low UHF range, with possible peaks-resonances at particular RF frequencies. The signal can be received with a Yagi antenna (10-12 dB gain), and amplified with a preamplifier (up to 20 dB gain), and fed into a standard analogue black-and-white TV receiver tuned to a resonant lobe. **Horizontal and vertical sync signals** (not received with the RF signal) need to be generated with **local signal generators and injected to the TV's synchronisation circuits**, to get a stable picture. Spacing between RF lobes will probably not match standard TV channels, so if the TV doesn't support continuous analogue tuning, it may be necessary to receive the signal using another precisely tuneable RF receiver, and convert the frequency - generate an IF by mixing with a local oscillator (LO), so the mixer output can be tuned to some standard TV channel.

Shielding reduces TEMPEST radiation, but can't eliminate it because every electronic device needs to have some open parts - ventilation ducts, connectors, and the screen surface itself must be visible. Putting many electronic devices together in a room was tried as a method of protection, but it didn't work, because they can transmit at different frequencies (explained previously). Filtering square waves is a good method. It may create fonts with blurred edges, but will significantly reduce TEMPEST radiation. Abrupt square pulses create strong higher harmonics, up to 100x original frequency.

Another interesting method of protection was proposed by Wim van Eck himself.

Normally, a PAL VDU picture is sequentially scanned by a single electron beam: first odd lines 1-3-5-...-625, and then even lines 2-4-...-624. The order of line numbers scanned can be shuffled in $625! = 6.14e+1477$ ways and the picture would remain the same for Alice. Not all combinations will make the picture fully unreadable for Eve, but her job will now be a lot more difficult. Scanning order can be even randomly changed in certain intervals. However, instead of a relatively slow linear ramp on vertical deflection plates in normal sequential scanning, the system will now have to use fast abrupt-shifting high voltage which may create yet another TEMPEST radiation channel!

The calculations here were made for a 1980s standard PAL VDU, displaying 25 frames / 50 fields per second at a typical 600x600 resolution. The situation is similar for any newer monitor, with a higher refresh rate (e.g. 150Hz non-interlaced⁹) and higher resolution (e.g. 2000x1000 pixels). Here it takes $6.67\mu\text{s}$ to scan 2000 pixels, which is only 3.3ns per pixel. This means that RF frequencies that will transmit the TEMPEST radiation will be cca. 30x higher, deep inside the microwave range (up to 15GHz). RF receivers for this range are more expensive and sophisticated, but on the other hand, here we have an even higher possibility of different conductor arrangements inside the VDU to behave like parasitic resonators - the lengths now required are only a few centimetres, and we don't need multiple turns to make an inductive element: all distributed inductors in this frequency range are straight unwound conductors (like wires or PCB traces), serving also as distributed capacitances in combination with other conductors.

Although modern colour LCD monitors use different signals to generate a picture, TEMPEST attack methods have been devised for them as well. The defensive method of "shuffling" when refreshing the pixels may work better here because LCD monitors don't operate on high voltages like their CRT predecessors.

Anyway, the work of Win van Eck caused a lot of security troubles for all sides involved in the Cold War, since every plain monochrome monitor (used in their thousands, all over every high-security NATO and WP installations) suddenly became a wireless listening device and a direct RF link with up to 1km of range, which is more than enough in every urban environment. As if this was not enough, many new TEMPEST attacks began to emerge...

3. **CRT visible light as a TEMPEST channel.** This channel is even more obvious than residual VHF-UHF, but was exploited much later. The first paper was published in 2002 by Markus Kuhn [10]. The device under attack was a standard RGB VGA CRT monitor, which was more advanced than an average 1980s VDU.

As the electron beam scans over phosphor particles (despite the name, they rarely contain the chemical element *phosphorous-P*) on a screen, at a certain intensity, proportional to **a video signal**, they emit visible light. The phosphor pixel visible light

⁹ **non-interlaced** means scanning each frame in only one field, while **interlaced** means two fields (odd and even) per each frame

emission will continue for more than 500ns after the electron beam has passed on to the next pixel in a row. The emission decay is similar to the step-response of a **linear first-order low-pass filter with a time constant of cca 100ns**, although a more precise mathematical model is needed to properly design a good corrective high-pass filter.

Each pixel emits a light pulse, less than 1 μ s long that will travel around the room where the monitor is located, bouncing off the walls. The intensity of light emitted by each pixel changes quickly as pixels are scanned, with higher-order frequency components up to 100MHz range on new monitors with higher resolutions and faster refresh rate. If captured by a **photosensitive detector**, and properly filtered, the image can be reconstructed even after a reflection of a wall, without having a screen directly in the line of sight! The electron beam scans each pixel sequentially, so a **single photo-sensor** is sufficient to capture the fast-changing reflected visible light intensity.

Photo-detectors with time constants of much less than 10ns must be used. **PIN photodiodes**¹⁰ can be made with a response time of less than 1ns, but won't work well here because of the too low sensitivity¹¹ of 0.5-1.0 A/W. **Avalanche**¹² **photodiodes** have a much higher sensitivity, typically 100A/W, with equally fast response times. **Photomultiplier**¹³ tubes have even higher sensitivity, up to 10⁶A/W.

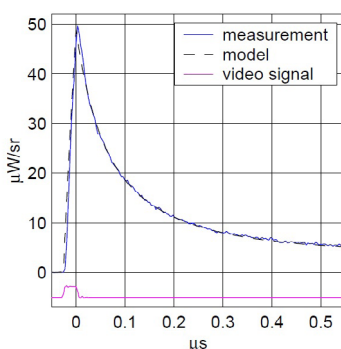


Figure 2.3 - A single blue pixel response, on a 640x480, 85Hz RGB monitor

A signal captured by a photo-detector needs to be filtered with a good corrective high-pass filter since a phosphor pixel behaves like a low-pass filter. Its output (intensity of emitted

10 A photodiode with un-doped intrinsic layer I between doped P and N layers - thickness of I layer increases a distance between P and N region in addition to depletion region, to decrease the junction capacitance in reverse bias, which decreases response time.

11 a ratio of photocurrent generated (in reverse voltage bias) to the power of incident light that depends also on the wavelength of light.

12 APD-a solid-state device. Uses a mechanism similar to avalanche breakdown in Zener diodes to multiply photocurrent-generated electrons. Requires a reverse bias >100V.

13 PMT-vacuum tube with a photo-cathode. Requires a much higher bias voltage than APD.

visible light) depends on input (electrical video signal to CRT monitor), or (in the other words) output signal is a consequence (or mathematical function) of the input signal.

As you can see in figure 2.4, the test image on the screen consists of texts of different font sizes, and colour samples.

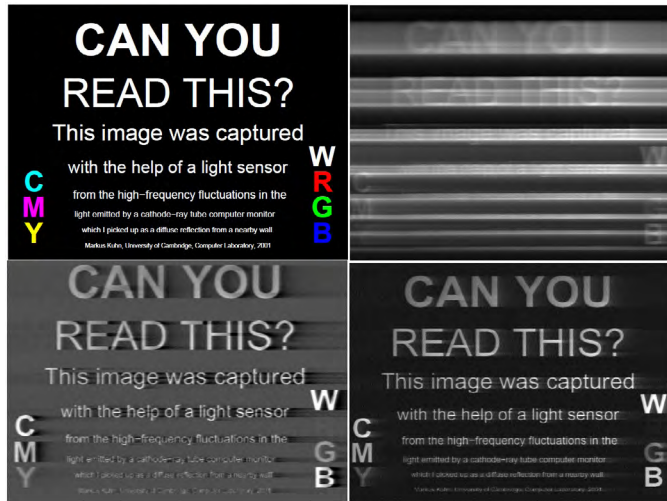


Figure 2.4 - left to right, the original picture, captured image without filtering, image processing with **1st order high-pass filter**, and image processing with advanced **deconvolution filter**. The PMT used had a higher sensitivity for blue than for red light.

Without additional filtering, the smeared image on the upper-right will be captured. This image requires additional filtering to remove the phosphor pixel response characteristic. Compensation with a simple **1st order high-pass filter** improves the quality. However, since the exact pixel response model is much more complex than a simple 1st order low-pass filter, a **deconvolution filter**¹⁴ can be designed based on that model, and an even better quality image can be achieved.

Similar methods of attack, modified for different frequency spans can also be used to attack for example LCD or OLED monitors because scanning and refreshing the image on them also requires multiplexing signals and serial transfer of data from a computer to a monitor, similar to CRT.

2.4.2 • How to defend on a DIY budget?

As you have just seen in these 3 historical examples, the equipment used was never cutting-edge ultra-high-tech for its time. Full defence against TEMPEST can be very complicated and expensive. Once again, let's take a DIY low-tech approach and see what Alice and Bob can do. There are many known and unknown TEMPEST channels, so let's try to reduce the threat and simply avoid using setups that are too difficult or expensive to defend.

14 Filter designed based on a discrete Fourier transform of a pixel's impulse response.

1. **Choosing a good location** - TEMPEST attacks are easier to mount in densely populated urban areas than rural, in big apartment blocks rather than single-detached houses and against static targets. High-gain, highly directional antennas are needed at longer distances, or to amplify weak signals coming through reinforced concrete walls (please note that newer, **higher-res monitors**, emit **higher frequencies** and hence require **smaller antennas** - easier for Eve to handle). They are easier to set up and conceal in big buildings. It is easier for Eve to approach closer to Alice in an urban area. It is easier to keep a directional antenna aimed at Alice if she stays in one place. Moving while operating helps Alice a lot.
2. **Shielding** - this is obvious, but often difficult to properly implement. Shielding an entire room is more difficult than shielding a single electronic device. Rooms and devices need to have apertures - for cables and ventilation in the first place. Coaxial cables can be fully shielded. It is difficult to fully shield keyboards and monitors. Regarding the frequencies involved, either lower or higher can have (dis)advantages. Higher frequencies (lower wavelengths) will more easily escape through small apertures. Again, from the microwave electronics textbook, shorter dipole antennas transmit higher frequencies better, and the same goes for shorter apertures, so-called **slot antennas**. There is a **duality principle** - a horizontal slot behaves like a vertical dipole (magnetic field- horizontal, electric field -vertical). On the other hand, lower frequencies have higher penetration depth into conductive material (see 2.4.1), and so require a thicker layer of metal shield to attenuate properly. Careful analysis of the residual RF frequencies transmitted is required first, so proper shielding can be designed, depending on what frequencies need to be attenuated. Measurements with a wide-band spectrum analyser are then required on a working prototype to confirm the results. The whole procedure is rarely performed and this includes large well-funded organisations and those directly involved in the security business. Therefore, this will be especially demanding and time-consuming for DIY spies Alice and Bob.
3. **Using lower frequencies** - it is easier to apply a thicker metal layer shield than plugging every millimetre-size aperture. Low RF frequencies need added parasitic antenna length to be effectively transmitted. Filtering **square waves** (waveform normally used by digital circuits, but not needed to carry information on long-distance) to **sine waves** helps a lot since it eliminates all higher frequency harmonics. Using fonts with blurred edges to display plaintext on a VDU works on this principle: blurred edge=slow signal.
4. **Use battery power if possible** - putting a battery inside a shielded metal box as a source of power eliminates the need for a power supply cable connection. Power supply cables can leak low-frequency TEMPEST radiation to the AC mains grid, where it can travel long distances (RF will be quickly attenuated on AC mains grid cables designed to carry 50/60 Hz). Furthermore, long AC mains cables can even behave as effective LF antennas, especially if suspended overhead. Even higher frequencies, in MHz range, can travel through AC mains grid far enough for Eve to intercept them in

Alice's neighbourhood - **Powerline modems**¹⁵ purposely designed to do this have been around for quite a while, and they are not particularly expensive these days.

Filtering a DC power supply, to fully block the complete RF span is difficult and requires at least a two-stage filter. For example, if a low-pass filter can block frequencies in a span 1-500MHz (designed e.g. after establishing that TEMPEST radiation can be expected at $f > 20\text{MHz}$), it doesn't mean that it will attenuate 3-10GHz as well - the **lumped component** low-pass filter that works well at $f < 1\text{GHz}$ may start to behave like a high-pass **distributed component** filter in the microwave range. One reason is, e.g. a lumped capacitor, designed to operate on frequencies up to 1GHz, will start to behave like an inductor (see figure 2.5.) on higher frequencies, and may have an erratic behaviour deep inside the microwave range (when all the unmatched transmission-line effects begin to take effect). This applies to other lumped passive components as well.

This means the lumped component low-pass filter must be followed in series with a properly designed microwave low-pass filter (using an appropriate microwave design method for a filter with distributed components- e.g. **a microstrip**¹⁶ filter). To confirm good filtering, extensive testing with a good RF spectrum analyzer is always mandatory.

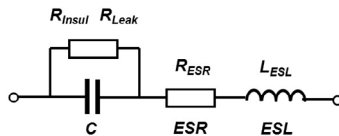


Figure 2.5 - a lumped capacitor's equivalent circuit for high frequencies

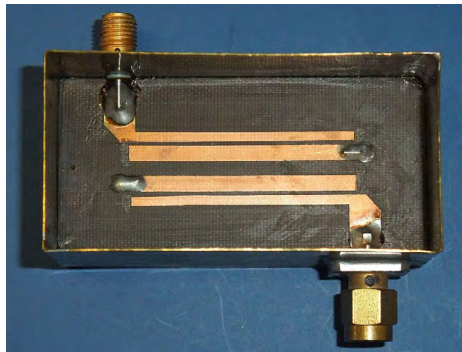


Figure 2.6 - an example of a microstrip microwave RF filter, with coaxial SMA connectors

¹⁵ modems designed to communicate through existing AC mains grid cables. Capable of operating in Mbit/s speeds on short distances up to 100 meters. The main problem of powerline communications are poorly matched impedances and big losses at MHz frequencies in mains cables and connections designed to carry only 50-60Hz.

¹⁶ a type of a microwave filter, realised on a two-sided PCB, with a full bottom copper ground plane and strips of copper on the top plane and a substrate dielectric in the middle. The microstrip behaves as a transmission line, with a substrate insulator and air as dielectrics.

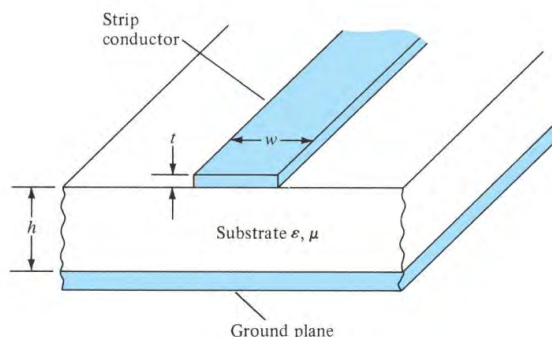


Figure 2.7 - a cross-section of a microstrip line

- 5. Local scrambling** - Alice's super-duper crypto system will transmit and receive **ciphertext long-distance** to and from Bob but inevitably needs to process and display the **plaintext locally**. Local scrambling is any procedure similar to the aforementioned shuffling of scanning the order of lines on a CRT VDU, to mitigate van Eck phreaking. The information from Alice's computer to her VDU is transmitted serially, so Eve needs to properly re-arrange the captured signals, to a properly timed sequence, to get meaningful information - the first action required to get a stable and legible picture was to re-generate horizontal and vertical sync signals. A similar procedure can be applied to other devices. If a keyboard's cable is the main source of a keyboard's TEMPEST radiation, then scrambling of bits on a PS/2 cable may be very useful. For this method to be effective, **Alice must first find what the main source of TEMPEST radiation on her devices is** - to know where to introduce scrambling. If the main TEMPEST is radiated on a length of a video signal cable between a PC and a VDU, and not inside the VDU itself, then it is useless to transfer a video signal in plaintext to VDU and then scramble it inside the VDU. Another example is a noisy dot-matrix printer - the main source of TEMPEST here is acoustic, not electrical. Electronic scrambling on a data cable doesn't help here. Printing each line of text with multiple passes of a printer head over the same line of text can help by shuffling the order of dots to be punched. Please note a different **pseudo-random** or even **true-random** dot-bit shuffling pattern can be used for every following line of text, with the same result: a legible message in plaintext printed on Alice's paper.
- 6. Jamming** - TEMPEST signals are weak. They are always well below levels defined by FCC¹⁷ rules. The FCC's main concern is to keep RF radiation below certain levels which may cause harmful interference to other devices. The designer and manufacturer's main concern is not to violate FCC rules. If it was, why not try jamming TEMPEST? Let's transmit jamming signals at 10x the amplitude of TEMPEST emanations, and they will mask the TEMPEST, while still staying below the maximum defined by the FCC.

The procedure is first to measure TEMPEST signals at different points around the device, using a spectrum analyzer. Then, observe the amplitude and frequency of

¹⁷ Federal Communications Commission - the USA government agency for regulating and enforcing communications laws.

TEMPEST signals followed by building a variable frequency RF oscillator, and sweeping the frequency over the spans of interest. Put the RF jammer transmitting antenna inside or around the device or simply use an unused pair of wires inside a signal cable. Use a good RNG (another example of how a good RNG is a pre-requisite for any serious crypto-system) to randomly shuffle both the amplitude and frequency of the jamming signal. If all the leaking signal RF bands are well covered, Eve will not be able to extract weak TEMPEST radiation signals from much stronger and unpredictable (if the RNG is good) jamming signals. The problem with this method is that TEMPEST signals may be present in a very wide band, from kHz to GHz. This all needs to be measured carefully, before proceeding to design the multi-band jammer itself.

7. **E-paper displays** - their extremely low power consumption for high endurance on batteries is not important here. What is important is that they need only a small amount of energy to refresh, and zero energy to maintain a still image, meaning **a very low residual radiation**. Eve will have a hard time intercepting a short and weak TEMPEST signal, present only while refreshing the image. They need several seconds to fully refresh one single frame, but this may still be OK for Alice and Bob to display, read and scroll the plaintext.
8. **A printer instead of a monitor** - this is how **teletypes**¹⁸ worked in the old days. Printers are much easier to fully protect using an aluminum RF shield than monitors. The frequencies involved to transfer plaintext bytes, without any graphics, are very low, meaning slow signals and hence low TEMPEST radiation. If housed inside an aluminum box, with only a slot for a paper tape (e.g. a 0.5mm wide slot, with 60mm length for a standard 57mm paper tape), will attenuate the RF below 2.5GHz. If this slot is extended to a **“waveguide”** with a 0.5x60mm cross-section, e.g. 10 cm long, this will attenuate the RF even better. The text on the paper output updates even faster than on an e-paper display.

Depending on the type, some printers generate more acoustic, some more electrical TEMPEST. **Thermal printers** are preferred, because they are relatively silent, and they also don't consume too much electrical power, which also means low electrical TEMPEST. Small 57mm paper tape models can be bought for 50EUR (our poor spies Alice and Bob love them). Dot-matrix or daisy-wheel printers create a lot of acoustic noise, which can be correlated to the characters being printed.

9. **Avoid using electronic devices** - how about preparing the ciphertext with one-time pads using pen and paper, and then sending the encrypted message electronically? Then receive the encrypted message from Bob electronically, display it on a VDU, and decode it manually on paper again? This way the plaintext remains **on paper all the time**, without a chance to leak any electronic or acoustic TEMPEST signal! It can be done using one-time pads since it is one mathematically extremely simple procedure.

18 Electromechanical communication terminals from the past, consisting of a keyboard and a printer (printing normal ASCII text) at every station. A wired or wireless link was used for connection, usually to a switched network (sometimes a standard public telephone exchange, switching the same as analogue landline telephones)

No need for EMI-RF shielding, or any anti-TEMPEST procedure at all! A true random number generator can still be used to generate random numbers for OTP keys since it is easy to put the RNG inside a well-shielded aluminum box and power it with batteries if required.

On the other hand, if Alice wants to send it using a Morse radio telegraph, she will do much better to use an electronic device to generate dots and dashes, since keying the Morse signals manually inevitably embeds a signature of Alice's hand. This example shows that although **low-tech almost always beats high-tech** in security, this rule is **not without exceptions!**

Electronic devices are most certainly useful in this world, but sometimes even 1000 years old methods work better against the high-tech Eve and Mallory. A few minutes of extra patience to do the critical work with pen and paper can create a difference between life and death for Alice. This is why Alice and Bob need to abandon this high-tech-as-quick-as-possible-let-the-computer-do-the-thinking-for-me mindset of the modern world, which may be very difficult - it is unfortunately already formed and widespread by a vast majority, including professionals with an electronic engineering background.

2.5 • Hardware Trojans

As I said before, we won't deal much with **software malware** in this book. It has been dealt with by many in the past and present, and many specialised software companies can provide defences. The best way Alice and Bob can do DIY is to build and use simple systems that can't be attacked using malicious software. Has anybody heard about a virus capable of successfully infecting a ZX Spectrum or Commodore 64 operating on cassette tapes?

Hardware Trojans are modifications that are physically installed/planted by Mallory to Alice's and Bob's devices on a hardware level. Let's suppose Alice and Bob will be guarding and hiding their devices well enough, so Mallory won't get a chance to fiddle with their fully assembled hardware devices. On the other hand, Mallory has a greater chance of success if she can manipulate individual components which are to be installed to Alice's crypto gear. Alice can trust the safety and security of her secret vaults to hide her components and operational devices. She can learn enough about electronics to design, program, and/or assemble all of her crypto gear herself. However, she unquestionably can't afford to manufacture the electronic components herself on a DIY budget. Even well-funded government organisations sometimes outsource the production of security-critical components to private companies, located in foreign countries, mainly because of lower prices: an extremely bad practice, of course. Alice will also have to buy her electronic components from untrustworthy sources, and Mallory can step in and plant rigged-manipulated components to Alice's mail shipped from Ebay, Ali Express or RS catalogue.

There are many ways for Mallory to steal Alice's secrets or disrupt her operations in another way. Mallory can try attacking Alice's random-number generator by planting a hardware Trojan (see 2.2.2). She can also plant a Trojan to Alice's encryption device to leak her

plaintext, or simply record it. Please note that the hardware Trojan attack method wasn't very effective in the 20th century, mainly because of the too high price of all the resources required. The integrated circuit design was a slow and expensive process back then, and technology wasn't advanced enough to tackle this effectively.

Nowadays, Mallory can easily design her version of an MCU that Alice likes using (e.g. ATmega328P), by adding extra circuits to its original silicon die and then give her blueprints to a Chinese factory to produce it for her. She can later stamp the chips to look exactly like originals from Atmel. She can easily add megabytes of flash or EEPROM memory (to an 8-bit MCU designed to operate on kilobytes), and extra circuitry to control the copying of data to the special part of non-volatile memory accessible only to her and Eve, and still, stay within a mid-range budget. The possibilities are endless.

2.5.1 • Types of hardware trojan

They can be designed to attack Alice's computer in many different ways. Let's start with a few words about the two basic types:

1. **General-purpose Trojans** - Although most CPUs operate with kilobytes of memory (simple 8-bit CPUs and MCUs that Alice and Bob will use - more advanced CPUs are faster but more complex and hence less secure - 8-bit CPUs are fast enough), nowadays even gigabytes can be added to an MCU's silicon die for a few extra dollars. GP Trojans operate in the same way, regardless of what the main CPU program does. They don't depend on firmware or electronic circuits where the CPU is installed.

They usually operate in a way that simply copies the contents of RAM to extra planted secret storage memory, accessible only by Eve. Copying is done in regular time intervals, or on some special events (e.g. activating a sleep mode). Since RAM (usually SRAM on MCUs, DRAM is less efficient in small systems) is volatile, the secrets are usually contained there. Eve can hack into flash or EEPROM (upon seizing some of Alice's devices), but RAM will be lost. No matter what the function of Alice's device is, frozen snapshots of RAM will always be useful to Eve.

The circuitry required to be planted to a silicon die is relatively simple. The secret controller must only eavesdrop on an address and data bus and grab data when SRAM is accessed. This way it won't interfere with the normal functions of Alice's device. However, it will pull extra current (extra milliamps at a duration of several milliseconds are required for writing to secret flash or EEPROM storage), which can be detected.

2. **Application-specific Trojans** - These are much more advanced. Instead of logging Alice's secret data, they are active against the operation of Alice's digital device. They will mostly serve Mallory, not Eve. They will, for example, try erasing various crucial protective actions from Alice's firmware (e.g. by filling the program code with NOPs or by inserting a JUMP). They may even try to transmit Bob's plaintext to Eve's IP address. The scope of possible actions is much wider than for general-purpose Trojans.

For them to work, they first need to properly identify the MCU's firmware, to 100% positively identify the device as e.g. Alice's OTP crypto teletype device - with false identification, the modifications they make inside the firmware are likely to cause easily detectable malfunction to Alice's device. Even if Alice doesn't suspect a Trojan, she will simply consider the MCU faulty (or even the whole batch ordered from e.g. Ebay, so she will choose a different supplier next time), and replace it. Mallory can make a database of all firmware used by Alice and Bob, and include it in the hardware Trojan installed to MCUs that she will plant to Alice's shipment from RS. Needless to say, since this type of attack requires more complex micro-circuitry, and more preparations (Mallory needs to know all Alice's firmware as well), it is more demanding and expensive and has much less chance of succeeding, than an attack with a general-purpose Trojan.

2.5.2 • East German Z80 clone vs. the newest 10nm FPGA

Let's begin a few decades earlier. It began with electromechanical relay computers, followed by analogue electronic computers, and then vacuum tube digital computers. Then came discreet transistor digital computers. All of them had one good trait (actually a very favourable one from the standpoint of security): they all required various **mechanical** actions for programming. This may have been flicking switches or pushing buttons, connecting wires from one panel to the other (a so-called **"hardwire" programming**), setting dials and potentiometers, etc. This means that they couldn't be remotely reprogrammed, or locally without a human operator's physical action.

Even if Mallory could get close enough to rig a hardware Trojan -i.e. to add some extra circuitry or change the existing one, it would be very easy to detect by Alice and Bob. The dimensions of components were in a metres to centimetres range: any change in hardware would be easy to spot with their naked eyes.

Using any one of these types of computers would be very impractical for Alice and Bob, however they don't need to go back this far in time to secure against hardware Trojans using retro technology. Going back to the beginning of the 1980s is enough. During this time (the Cold War), the world of electronic design and manufacturing was quite different to today. Almost every mid-developed country (including Yugoslavia) had an electronic industry, both in component production and device production level, unlike today, where East Asia (mainly China) produces most of the world's electronic components. Every country actively involved in the Cold War, on the East or West, needed to develop, design, and produce electronic technology itself, so it could rely on its industry in case the Cold War heated up. Both blocs have often relied on stealing blueprints, reverse engineering, and eventually producing illegally cloned devices in factories based inside their territory. This is sometimes difficult to explain to younger generations who have grown up in a world where China produces almost all of the world's cheap electronics. Electronic design engineering was much slower, resulting in increased reliability and vigorously tested products. Much fewer types of CPU and other digital ICs were available. The few types of CPUs that were available were used in a very wide range of applications, from home computers and gaming consoles, to industrial process automation and laboratory instrumentation. This means that for example, the Zilog Z80 has undergone extensive testing in all sorts of applications, in

every possible environment (desert, jungle, underwater, North and South pole, even deep space,...) from its introduction in the mid 1970s.

This simply means that you can trust it more than any new cutting-edge high tech - also thanks to its low level of integration, and also the absence of EEPROM and flash memory (Alice and Bob will do fine using a lot more secure UV EPROM - all explained in 1.1.3 and 1.3). There is also another reason: The Z80 and 27C UV EPROMs were used actively for almost 20 years. They were produced in their millions, and scattered all over the globe and hence easily available to Alice and Bob. If Mallory wants to rig them with a type of hardware Trojan, it will be very hard for her, since they are available in millions.

It may be even impossible if Alice has kept a stash in a secure vault since 1980s - which is not so improbable scenario. Mallory could easily rig a Z80 or a 27C UVEPROM today. It is almost impossible the STASI did it back in the 1980s with Z80 clones¹⁹, reverse-engineered and produced in East Germany. Although EG was keeping up very well in the high-tech and arms race, such a project would be way too expensive to pull off using 1980s technology. If Bob had a chance to get a stash of Ux880s and UB857s from the Erfurt factory - which was very easy during the times of the collapse of communism in 1989-1990, he would be equipped with 100% Trojan-free certified technology! **Please note that this is very difficult, if not impossible to pull off today!** In 1990. we were all (including me, as well) mainly still concerned with how to smuggle brand new high-tech contraband²⁰ from the West, still completely neglecting the electronic security aspect - who would have thought of a possibility of exploiting comrade Honecker's technology back then - to **increase your security!**

Modern, highly integrated components like FPGAs or specialised DSP CPUs, are very insecure by design and are not to be considered for building secure hardware for Alice and Bob. Furthermore, a Z80 running on 4MHz, equipped with 64KB of memory is fast and powerful enough for anything that Alice and Bob might need. We will see it in the following chapters as we proceed to design highly secure devices for our poor, low-budget DIY spies. Let's just make another obvious remark regarding very popular **softcore**²¹ contemporary **microprocessors**. They are a nightmare from the standpoint of security. They not only enable Mallory to change the program to suit her needs, but also the internal CPU core circuitry (and hence the CPU machine instruction set) if she feels like it.

19 Ux880D: a copy of Zilog Z80 CPU by VEB Mikroelektronik "Karl Marx" from Erfurt. Besides the CPU, the other Z80 system components were also cloned in East Germany, e.g. UB857D-a good copy of Z80-CTC (a 4-channel counter-timer IC - an add-on to Z80 CPU to relieve it of simple counting and timing tasks, and hence increase the precision and speed)

20 contraband: a term describing illegal, smuggled items - e.g. a ZX Spectrum inside Yugoslavia in 1980s or, even worse, Cuban cigars inside USA after 1959.

21 softcore CPU: usually realised using an FPGA. A so called hardware description language (like VHDL or Verilog) is first used to define, arrange and interconnect all the internal logic gates and other digital logic elements (using software!). This actually defines the CPU core and also its machine instruction set at a low level. The software-defined CPU core can then be programmed using standard programming languages (like assembler or C). Offers great flexibility, equal to redesigning the CPU core each time as needed. They are also popular for fully emulating old computers by constructing their CPU cores by the means of VHDL software for retro gaming.

You can see in figure 2.8. that Z80 components from Erfurt are very easy to recognise by the DIP package design (a notch at the upper left corner).



Figure 2.8 - Contraband from the East: unlicensed reverse-engineered Zilog Z80 clones: Z80 CPU-UB880D, Z80 SIO-UB8560D, Z80 PIO-UB855D, and Z80 CTC-UB857D

2.5.3 • Planting, detecting, and countermeasures

Methods of planting hardware Trojans have evolved. The simplest method is to add extra circuitry to a silicon die **beside** the original in the same plane. This can be detected after decapsulation, with a good microscope. Another method is to hide the extra circuitry **under** the original, which is more difficult for Mallory to perform, but also more problematical to detect by Alice's microscope. According to [14], it is possible to add backdoors even **without changing the microcircuits**. Just manipulating dopant levels is enough!. This way, microcircuits, even in 3D, will look the same as the originals.

The problem for Alice and Bob is that Eve and Mallory always have access to much superior technology. Microscope and decapsulation equipment is probably the most advanced technology that they would ever have. The good news for them is that they can mitigate the hardware Trojans with much lower-level technology. Let's go through some options.

1. **Timing:** This technique is mentioned before, in 2.1. Here it will be used to perform a simpler task. A hardware Trojan may cause extra CPU load as it might have to spend extra CPU cycles. Alice can monitor a suspicious IC compared to one **"golden chip"**.²² If the IC under test starts lagging behind the golden chip when performing the same task, it may have a hardware Trojan. Sometimes, the Trojan may slow the CPU down to a level where it may malfunction in performing its usual tasks. It then becomes very easy to detect.
2. **Power analysis:** also explained in 2.1. An extra circuit will consume extra power, so a comparison to a golden chip will reveal extra power consumption when the Trojan kicks in. It is important to check that both DUTs²³ are made in the same variant of technology

²² golden chip: a reference IC for which Alice knows that it hasn't been tampered with, kept inside her highly secure vault. A good example is the UB880 fallen of a truck in 1990.

²³ device under test

(e.g. some Z80s were made in NMOS, some in CMOS technology which consumes less power than NMOS).

If a hardware Trojan is used to log the SRAM snapshots to extra planted EEPROM or flash, and operates in parallel with the main CPU, without consuming its CPU cycles, Alice can detect it by measuring the supply current with a digital oscilloscope. EEPROM and flash need a **few extra milliamps** for a **handful of milliseconds** to write a block of data, and this spike is easy to detect. Once the current spike is measured, she can modify her crypto-device to stop or enter a special ISR²⁴ or otherwise simply light up a red LED to alarm her and Bob each time the supply current rises above a certain threshold for several milliseconds.

Besides EEPROM or flash, there are other technologies that Mallory can use for on-chip non-volatile memory that consume much less current and are difficult to detect. One of them is e.g. FERAM²⁵. Its endurance is more than 10^{10} write cycles (much more than 10^5 - 10^6 for EEPROM) and data retention is better than EEPROM (for several decades). The disadvantages are their relatively high price and low level of integration - 1MB is the largest memory chip commercial available. Furthermore, a FERAM cell needs a layer of PZT²⁶, and also isn't available with a track width less than 100nm, which may be difficult to integrate with some newer CPUs.

3. **Using low-tech:** all the advantages of low-level integration, ICs kept in secure vaults since the 1980s, with a separate CPU, SRAM, and UV EPROM (impossible to reprogram with low voltage in runtime) were explained already. Using additional **hardwire programming** (see 2.5.2) for configuration of various critical parts is also recommended because it can't be altered by a Trojan. We will get into more detail in chapter 5.

Here I will point out another type of hardware Trojan: a so-called **CPU instruction level backdoor**. This is about introducing new assembly-level commands or opcodes. Mallory plants extra logic gates circuits to a CPU instruction decoder. This introduces new opcodes that can be decoded and executed. Some CPUs will not execute every possible sequence of 8-bit values that come to their instruction decoder input. The **illegal opcodes**²⁷ will be rejected and may activate a special ISR, called a **trap**. Unlike other CPUs, the **Z80 doesn't have a trap**. Every combination of hex bytes brought to its instruction decoder will execute some correct operation [8]. It does have some

24 interrupt service routine

25 FerroElectric RAM: a low-power non-volatile memory using a material with ferroelectric effect -i.e. a non-constant, non-linear, hysteresis characteristic of electric permittivity ϵ , similar to magnetic permeability μ of ferromagnetic materials.

26 lead-zirconate-titanate: a common ferroelectric material

27 illegal opcode: may refer to incorrect opcodes, impossible to decode by a CPU's instruction decoder. Also refers to opcodes actually possible to decode and perform some correct operation, but not listed in an official datasheet (so called "**undocumented opcodes**"). Sometimes accidentally omitted from the list, but sometimes intentionally left out - when manufacturer can't guarantee that these opcodes will perform well within a full span of defined operating conditions (like temperature or supply voltage).

undocumented instructions though, which have been extensively used and not reported (to my knowledge) to cause any instability. Since all combinations of multiple bytes (0x00-0xFF) can be decoded as correct instructions, there is no way for Mallory to plant a CPU instruction level backdoor. All combinations of instruction bytes are “used up”. The low-tech Z80 wins again.

4. **Obfuscated code:** Application-specific hardware Trojans need to recognise the original program or they can't work. Mallory can't know if Alice will use this exact ATmega328 for her OTP Crypto device or automatic garden watering system. She can plant 20 pieces of ATmega328 ordered from RS to her address, all rigged with application-specific Trojans which will make plaintext leak from her OTP Crypto. The Trojan needs to recognise that the firmware loaded is for OTP Crypto, not for gardening or anything else before it makes harmful mods. If it modifies the watering controller firmware (as if it were OTP Crypto), it will malfunction and Alice would immediately notice, and probably discard the whole batch of MCUs. Mallory's attack would thus fail.

If 8KB of flash program memory is enough to run OTP Crypto, Alice can fill the remaining 24KB with completely useless random garbage code never to be executed. Furthermore, she can split the 8KB of code into 10-20 sections and scatter them all over the 32KB flash memory. The sections can be cross-linked in a zig-zag fashion. She can also make 500-1000 sections.

To generate the garbage code, a good random number generator will come in handy here. Alice can write a special program that will take the 8KB of firmware (already compiled to assembly code), shuffle it, cross-link (again in a random way, based on RNG output), and fill the rest with garbage code. Each OTP Crypto MCU can then be loaded with a unique variant of obfuscated firmware performing the same tasks, but looking much different to Mallory's Trojan, which won't be able to correctly patch the code.

5. **Time-stamping:** This method can be useful if Mallory somehow manages to physically get a hold of Alice's electronic device containing electronic components previously rigged with a general-purpose hardware Trojan (e.g. **with a little help from Trudy**²⁸ who stole the device from Alice). She then gives the IC to Eve to analyse the recorded data logged from RAM. Eve may not exactly know how much time has elapsed since the last snapshot of Alice's SRAM was taken by the Trojan. After reading the SRAM snapshot logs created by the Trojan, she can revert the SRAM and everything else to the state in the last recorded snapshot. She can restart the system and let it continue running and then give it to Trudy to return to Alice's base. Perhaps Alice and Bob won't notice anything.

28 Trudy the intruder. A crypto-character with different definitions, but here we will consider her an insider, with authorized access to some of Alice's and Bob's equipment, but spying for Eve and Mallory. These two operate from outside, and can't directly access Alice's secrets. Depending on a level of access Trudy has, her attacks may be extremely difficult to defend. The most prominent example of “Trudy” was Kim Philby.

If Alice programmed a timer (or stopwatch/software RTC²⁹) that kept running while the system was powered up, Bob will detect the tampering. Eve will take some time to read the data logged by a Trojan, and then revert the SRAM. The timer (also recorded in SRAM) will then be **delayed** by that amount of time.

Eve needs to know exactly where in SRAM the timer registers are to precisely advance it before restarting the system, so it isn't noticed by Bob. Alice can again use the obfuscated code technique to conceal the timer. Even better, she can use some kind of a simple **blockchain**. For example, she can use the **hash**³⁰ function SHA-1. Alice chooses a random 20-byte (160-bit) value x_0 to start the blockchain. Each second, the CPU calculates a new 20-byte value based on the old one (e.g. $x_{k+1} := \text{SHA1}(x_k \oplus C)$, where C is for example a constant 5-byte value). Every result produced by SHA-1 is a 160-bit number, while 35 bits is more than enough to count seconds in 1000 years. This means that every subsequent SHA-1 result will correspond to a specific number of seconds elapsed from the start (although SHA-1 can output 2^{160} different values, and 2^{35} is more than needed, it is still theoretically possible to get one of the previous values as an output and thus circle back, and remain circling back. This can be tested by Alice by pre-computing a long blockchain for a given initial random value x_0 , so Alice and Bob can check the correct timing this way. Computation of one SHA-1 value for a 20-byte input will take less than $1\mu\text{s}$, meaning Alice can pre-compute the blockchain for the next 10 years in less than 6 minutes (will consume cca. 6GB of memory) on her secure computer. If Eve wants to subvert this, she will need to analyse all details of Alice's firmware to compute the correct blockchain values in advance and correctly set the "blockchain timer".

2.6 • Exploiting inherently insecure physical properties

The previous section was about deliberately modifying Alice's hardware to steal secret data. But does Mallory need to bother with all that? These attacks are technically sophisticated and require lots of resources. They can be detected and Alice and Bob will then be alerted. They may relocate and change their identities, modus operandi, and methods of communication and encryption. After reading [13] you will see that most electronic components exhibit some kind of data remanence. Fully deleting the data is more difficult than it may seem at first. People are usually more concerned with securely saving the data than with securely deleting it. Remember the old days (the 1980s or even 1990s) when UPS³¹ or auto-save features were not so widespread, and hence forgetting to save a few hours of coding on your ZX Spectrum to a cassette tape could have been very frustrating in case of a sudden mains power outage late at night. Most commercially obtainable electronic components were not designed with any security in mind, and the same goes for electronic devices, systems, and even networks. Implementing security features is expensive and seldom

29 real-time clock

30 hash functions: non-linear, difficult to invert mathematical functions, extensively used in cryptography. They produce a fixed-length result (160 bits for SHA-1) from an input of arbitrary length. Calculating a hash function is relatively fast, but inverting it (calculating the input from the output) is very difficult and slow. Changing one bit in input will cause an unpredictably large change in the output result. Adjusting the input to produce the desired output is also very difficult - for a good hash function.

31 uninterrupted power supply

cost-effective.

Maybe Eve needs to exploit the inherent pitfalls of certain technology, on any level (component, device, system, or network level), to extract secret data, without any need for help from Mallory. This kind of attack is harder to detect and requires fewer resources. Here we will see what can be done on a component or device level.

2.6.1 • Deleting HDD and SSD devices

As mentioned in 1.3, the only electronic method to surely delete data (i.e. to zeroise) from a HDD or SSD would be to repeatedly access every sector and fill it with sequences of zeroes, sequences of ones, and repeat the process several times (even up to 10x, as required by the strictest standards). This was possible on old CHS HDD drives but became difficult on newer LBA HDD drives, and next to impossible on SSD drives (including SD cards, USB flash pen-drives, anything based on solid-state, programmable, and erasable with low voltage). Heating to a very high temperature is possible but destructive and not very practical. Other destructive methods, like high-voltage shocks, have also been tried, with poor results. This is all a consequence of the inherent properties of this technology (magnetic or solid-state storage) and not any special features deliberately introduced by designers or Mallory.

Imagine design engineers decide to make HDDs and SSDs with enhanced security features in the future, like an added option to directly access every single physical sector to enable zeroisation, overriding the usual **LBA and wear-levelling³² controllers**. This is not unaffordable to realise, it just requires some extra coding on the controller firmware. It may look nice, but the black-box problem arises again. How can Alice and Bob trust them? A good solution for Alice would be to try to implement it herself, e.g. by modding a disc controller (or even designing and programming a new one from scratch) of an existing HDD. This is a complex project, but it pays back. Alice will end up with a secure HDD that she will be able to partially or fully zeroise without destruction. Modding an SSD will without doubt require more effort.

Although well known, the disposal of old HDDs or SSDs is routinely done without due consideration (and also the disposal of printed papers containing all sort of critical data, much easier for Eve to read out), even by companies directly involved in security. Imagine the following scenario: Trudy can't gain access to Alice's HDD, but can damage the computer by for example causing a mains voltage surge. Alice has surely made regular backups (e.g. to a network drive also not accessible to Trudy), so she didn't lose data. She simply throws the old damaged computer into a dumpster and installs a new one. Eve later goes dumpster-diving (actually legal in many jurisdictions, as long as the dumpsters are located in public areas - Eve may risk only a small monetary fine for littering) and manages to recover the data from the HDD. Quick, simple, and not so easy to detect. Wearing a proper disguise, Eve can always pass off as a homeless person looking for recyclable items to make

32 Internal MCUs present on all the SSD devices - individual flash sectors can handle only a limited number of re-writes (typically 10.000) before they wear out. The MCU, among other tasks, handles rerouting of data between physical sectors as needed.

a few bucks. No expensive high-tech or complicated procedures required.

2.6.2 • Recovering data from old (E)EPROMs

We will start with an analysis of this routinely ignored security pitfall, by going through the basics of MOSFET transistors and EPROM memory. Figure 2.9 shows the evolution of an EPROM cell by adding an extra polysilicon **floating gate** between the usual MOSFET **control gate** (also called **select gate**) and a p-substrate. The floating gate has no electrical connections and is just an island of conductive polysilicon surrounded by SiO_2 dielectric insulation.

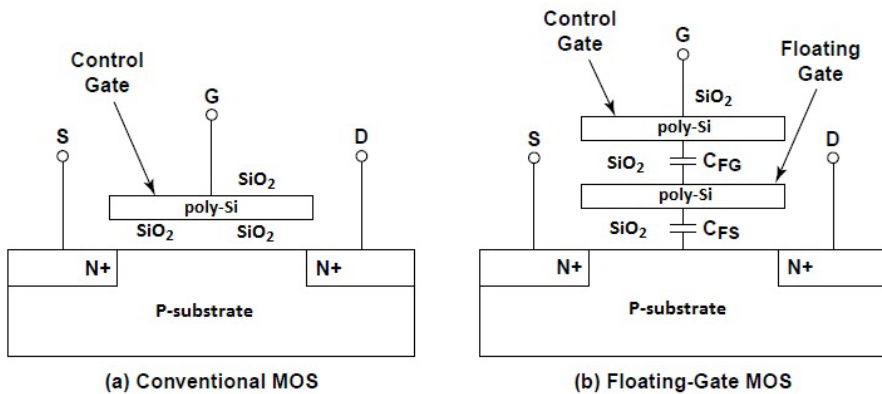


Figure 2.9 - A cross-section of a normal MOSFET and 1-bit EPROM cell

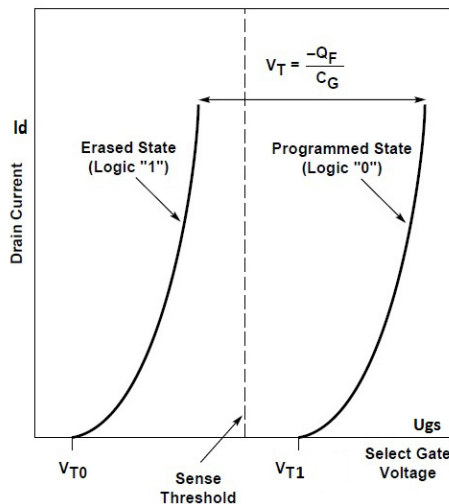


Figure 2.10 - Drain current I_D vs. gate voltage U_{GS} plot

An **enhancement-mode** (positive voltage on control gate is required to start conducting current between drain and source, by attracting electrons to the p-substrate surface) MOSFET is used. MOSFETs behave similar to a triode valve tube, i.e. as a voltage-controlled (U_{GS}) current source (I_D -flowing from drain to source direction). Gate current (I_G) is present while charging the gate capacitance, otherwise, it is extremely low. The following equation describes how U_{GS} controls the I_D of an enhancement-mode n-channel MOSFET, given that U_{DS} is sufficiently high (hence operating as a current source):

$$I_D = K(U_{GS} - V_{T0})^2, \text{ if } U_{GS} > V_{T0}, I_D = 0 \text{ otherwise, regardless of } U_{DS},$$

For a low $U_{DS} < 0.1V$, it behaves more like a voltage-controlled resistor:

$$I_D = \frac{U_{DS}}{R(U_{GS})}, R \text{ is a non-linear function of } U_{GS}, R \rightarrow \infty \text{ for } U_{GS} < V_{T0}$$

More precise equations for describing the MOSFET operation can be derived, but these are good enough to explain the operation of an EPROM memory cell.

In a so-called **erased** state (initial, or also called **unprogrammed** state) a standard EPROM cell will behave exactly like a MOSFET (with $V_{T0}=1V$), for gate voltages up to 5V (with a standard power supply voltage $V_{DD}=5V$). It will conduct several mA at $U_{GS}=5V$, and this is sensed as a logic "1". A transfer to a so-called **programmed** state is done electrically, by applying a much higher $U_{GS}=12-14V$ (a so-called "programming voltage") to the control gate (for typically 1ms). This way the electrons (**hot carriers**³³) pass through the SiO_2 layer and reach the floating gate, where they get **trapped in the polysilicon**. After the programming voltage is removed from the control gate, depending on temperature, they can stay there for 20-30 years. Since the floating gate is now negatively charged (a negative charge of electrons trapped at the floating gate $-Q_F$, on capacitance $C_G = C_{FG} + C_{FS}$, so now $V_{T1} = V_{T0} + Q_F/C_G$), a much higher positive U_{GS} is required to turn the MOSFET on (to start conducting), now with $V_{T1}=7V$. This means that the MOSFET will now not conduct (will be turned off) at $U_{GS}=5V$, and this is called a **programmed** state - sensed as a **logic "0"**. A transfer back to the **erased** (logic "1") state at UV EPROM is done by exposing the silicon die to UV light for 10-20 minutes. This will discharge all floating electrodes of all memory cells on the entire EPROM chip. Transfer to an erased state on EEPROM and flash memory is done electrically.

This EPROM cell (the same also applies to EEPROM, flash and all other variants of non-volatile memory), like everything else in this world, can't last forever and will surely start to fail after some time due to old age, or too much wear-and-tear. They can withstand only a limited number of program-erase cycles (typically 10000). This IC will then need to be replaced, and this is where security problems begin. Identified as non-functional, it will probably be simply **tossed to a trashcan** and then end up in a dumpster (it simply won't

³³ hot carriers: electrons with kinetic energy (because of their speed, which is proportional to electric field) much higher than their mean thermal energy at actual operating temperature. They are called "**hot**" because their kinetic energy corresponds to thermal energy at much higher temperatures than actual operating temperature. They pack enough energy to penetrate through the SiO_2 layer.

work anymore - a useless piece of electronic rubbish) just waiting to be picked up by Eve. Now we will analyse the possible failures and see, that using only **very low-tech gear**, Eve will stand at least an 80% chance to recover **all data** from the EPROM IC, although it can't function anymore inside Alice's crypto device under normal operating conditions.

1. **Hot carrier effects** - when programming the EPROM with "high voltage" (here meaning 12-14V, while "low voltage" refers to 0-5V), the hot carriers need to be generated for the programming process to function. Ideally, **all hot electrons** will be trapped at the poly-Si floating gate (when **programming**), and will **all** return to a substrate (when **erasing** with UV light). In the real world, some hot electrons will always be trapped in the SiO_2 insulator at many different spots and stay there, after repeated programming and erasing. This will gradually change the V_{T0} and V_{T1} threshold voltages, of course, not equally on every EPROM cell (a 0/1 bit of memory). One of the reasons why this happens is also because window-lens on UV EPROM are not perfect and will not pass the UV light to every part of the silicon die equally.

The effect is similar, in a way, to the charge-discharge cycle of for example a lead-acid battery. While discharging, **lead sulphate** (PbSO_4) collects both on positive and negative plate. While charging it converts to **lead dioxide** (PbO_2) on the positive plate and elementary **atomic lead** (Pb) on the negative plate. An ideal battery could be charged and discharged in an infinite number of cycles. In practice, the insoluble PbSO_4 will never perfectly stick to lead plates. A small portion will always fall off and sink to the bottom of the cell. These pieces are lost and can't participate in the charge-discharge reactions any longer, resulting in lead gradually being eaten away from the plates, and acid being diluted, until the end of the battery's service life.

The situation is worse with EEPROM where electrical pulses are used for erasing purposes. Unlike UV EPROM, where UV light exposure for too long (like 30-40 minutes) will not decrease V_{T0} lower than 1V, with EEPROM, a too-long electrical erase pulse can decrease the V_{T0} further, even below 0V, so the cell will start behaving like a **depletion-mode** MOSFET (starts conducting already at $V_{GS}=0$). It is much easier to generate large discrepancies between individual EEPROM cells (than between UV EPROM cells), that accumulate over time and eventually result in failure. **Flash** memory is far the worst in this respect. It electrically erases data in **blocks** (or **sectors** - typically 512 bytes), which also contributes to cell failures (no possibility to fine-control erase pulses for each bit or byte). This is why flash memory devices always have an MCU that takes care of marking the bad sectors and routing data to other sectors dynamically during normal operation. After some time, as soon as only 1 memory bit reads incorrectly when read during a normal run-time operation inside a crypto device, Alice may notice a malfunction, and eventually, replace the (E)EPROM IC.

According to [13], besides programming and erasing, even normal 5V voltage during operation in run-time will cause the hot electrons to gradually collect inside SiO_2 over a long period. This is because, at a given temperature and electric field, not all electrons will be at the same energy level. This is a matter of the **statistical distribution** of the thermal energies of electrons at a given temperature. Even at 20°C and 5V (and

even at constant voltage, the electric field E vector will not be the same at every point in space, because of the highly non-homogenous internal 3-D geometry of cells), very few electrons will, statistically, have enough energy to penetrate through SiO_2 . Please note this effect is **highly non-linear, related both to temperature and voltage**. Programming a bit (setting it to zero) with $U_{\text{GS}}=14\text{V}$ takes 1ms, but at least 10 years is required with $U_{\text{GS}}=5\text{V}$. This is very important to remember because we will later apply this to SRAM and practically every other MOSFET structure.

As microcircuit track widths decrease, and consequently, their distances in between, electric fields are increasing (in the simplest scalar case, $E=U/d$, but E is a 3-D vector, while U is scalar). Furthermore, since more complex 3-D structures are being packed together tighter, the effects of non-homogenous electric fields are increasing even more. Since 1970, track widths and distances have shrunk from $10\mu\text{m}$ to 10nm range, while operating voltages have decreased from 5V down to 1V - only 5x decrease in voltage for 1000x decrease in distances. Do the basic math now and tell me what has it done to electrical field vectors inside an IC. **Low tech wins again!**

2. **UV light leakage** - the glass window needs to be covered by some opaque tape (e.g. aluminium foil covered by black electrical tape works OK) after programming. If the cover is not good (I've seen many faulty UV EPROMs covered with a white paper sticker only), the UV light will leak inside slowly, and the bits will start to erase after several years. Please note high temperature, strong electrical and/or magnetic fields or any form of radiation present in the device's operating environment will accelerate this effect.
3. **Electro-migration** - usually a problem for power amplifiers, but may also cause problems with EPROM ICs. The impact of fast electrons may energise metal atoms (of metallic IC contacts) enough to dislodge them from their metallic crystal lattice, leaving behind space. After years of operation, this may slowly eat through the thin metal contacts and eventually break them.
4. **Overvoltage surge, excessive heat, fire**, or any other cause, usually generated by other parts of the system (like a power supply), which will destroy the EPROM IC in a matter of seconds.

Some of these failures (or their variants) will cause the EPROM IC to end up in the trashcan. Now let's see what Eve can do to extract all crucial data back in her lab, after a successful dumpster-diving session.

I learned these methods well, in my first job. I did PLC programming and commissioning at a small company that handled automation and remote control for a public water supply company on the island of Krk, in the north Adriatic. Old electronic control systems, installed more than 25 years before started to fail, mainly because of the 27C UV EPROMs as described. Not all backups were kept, so I had to find a way to extract data from faulty ICs. I found 27Cs that failed due to any one of the 4 aforementioned reasons, and each required a different approach. Sorry to disappoint you, there wasn't any security-critical information

within these 27Cs.

1. **Hot carrier damage:** As previously mentioned, negative electron charge will build up gradually after 25+ years of operation, usually inside silicon dioxide insulator. Each time a cell memory is read by a CPU, 5V is applied to the control gate. Statistically, very few electrons will have enough energy to penetrate SiO_2 at 5V (please remember that higher temperature increases this effect), and even fewer to reach the poly-Si floating gate, but the excess charge will slowly build up. This excess negative charge will slowly increase both V_{T0} and V_{T1} . Nothing will happen to cells in the logic “0” state. If V_{T1} increases from 7V to 9V, there will still be zero I_D at $U_{GS}=5V$. On the other hand, if V_{T0} increases from 1V to 3V, the drain current I_D at $U_{GS}=5V$ will become too low, and the bit will be misread as “0”. If this was a low-order bit in a byte of some mathematical constant or fixed analogue setpoint, it may not be immediately noticed. If this byte was an important opcode instruction (this may happen both on von Neumann’s or Harvard’s computers!), it will be noticed when for example opcode 0x77 is read as 0x76. To Zilog Z80 CPU it means a “HALT” instruction will be executed instead of “LD (HL), A”. The runaway code condition will be noticed immediately, but its exact cause may be very difficult to pinpoint.

If Eve gets a hold of this faulty 27C, she can easily recover all the data! All she needs is an EPROM programmer with a variable power supply voltage V_{DD} (worked just fine for me). If the program went runaway because the V_{T0} had increased to a point where the bit “1” was misread as “0” at $V_{DD}=5V$, if read with $V_{DD}=6.5V$ and hence $U_{GS}=6.5V$ (these 27Cs can handle $V_{DD} < 7V$), it will be read correctly as “1”! If it took 20 years for V_{T0} to increase by 2V, then V_{DD} and U_{GS} increased by 1.5V will compensate (because $U_{GS}-V_{T0}$ is what counts) the last 15 years of hot carrier frenzy. Since V_{T1} is also increased (now at $V_{T1} > 8V$), the cell with bit “0” will still be correctly read at $V_{DD}=6.5V$, will not start to conduct. Eve will do a few successive reads at $V_{DD}=6.5V$ and compare the results. She will then decrease V_{DD} to 6.0V, 5.5V, 5.2V, 5.0V, 4.8V and read again. The results will still match at 6.0V and 5.5V, but at $< 5.2V$ they will probably differ, which is consistent with hot carrier damage failure. She will then try to locate CRC checksums inside the firmware for extra precaution, and see if they match. Then she will analyse the assembler opcodes and see if they make sense (a check for illegal opcodes). Subsequently, she will consider the readout to be successful, securely destroy the 27C and save the data. In my case, once I was assured the firmware was correctly read, I discarded the old 27C and burned the firmware to a new one, and the pump station was ready to resume operation.

In normal operating conditions (temperature, voltage, etc.) and the case of UV EPROMs, **if the lens is covered properly**, it is the most likely fault to occur. More than 60% of faulty UV EPROMs that I processed had this fault, so I recovered the data by reading with **increased V_{DD}** .

2. **UV light leakage damage:** Unlike hot carriers that gradually increase the negative gate charge, the leakage of UV light will decrease negative charge. This means nothing will happen to cells in a logic “1” state. The floating electrode had zero electron charge

on it, so extra UV light can't remove anymore. Remember 2x or 3x more exposure to a UV lamp will not harm the UV EPROM when erasing.

The cells in logic state "0" will probably start to conduct and be misread as "1" at $V_{DD}=5V$ and $U_{GS}=5V$ when UV light leakage decreases the V_{T1} down to 4V by removing some of the electron charge at the floating gate electrode. This will cause the CPU to malfunction similarly to hot carrier damage failure.

If Eve gets hold of this 27C, she has to do the opposite. Now the V_{DD} of the EPROM programmer power supply will be decreased to 4.5V or 4.0V. This way the cell in logic state "0" will not conduct (because $V_{T1}=4.0V$) and will read correctly as "0". The cell in logic state "1" ($V_{T0}=1.0V$) will still be read correctly as "1" because it will sufficiently conduct.

The rest of the procedure is similar to hot carrier damage. Since the UV EPROM IC didn't exhibit any damage, it doesn't have to be replaced with a new one. Maybe a single erasing with a UV lamp and then programming the firmware again will be enough to refresh it properly.

Another 20% of the faulty UV EPROMs I managed to read and recover the firmware had this kind of fault. Almost none of them had their window-lens covered properly. White paper stickers don't block UV light well.

3. **Electro-migration damage:** This kind of damage includes broken metallic contacts. Decapsulating the IC and inserting micro-probes to bridge the broken contacts is what is needed to extract data from IC with this kind of damage. Decapsulation can be done using DIY means (a microscope and chemicals are needed), but **micro-probing**³⁴ requires expensive equipment. This kind of damage accounts for some 10% of failed UV EPROMs that I was unable to read.
4. **Over-voltage or other high-energy damage:** The rest were damaged in this way, which doesn't leave many practical possibilities for repair, and may require very expensive lab equipment to recover the data. On the other hand, during the experiments on the possibility of quick zeroisation of SSD drives, it was recognised that high-voltage shocks usually destroy controller electronics and connecting contacts, while the flash memory itself survives. After cleaning burnt plastic, ashes, and other debris and the decapsulation of silicon dies, a lot of micro-probing with a good microscope may still recover the data.

2.6.3 • SRAM and DRAM data remanence

Although these are volatile memories, they have a kind of residual retention. They were designed for speed, low price, density of data storage, reliability, etc, but not to be volatile by design requirements! Non-volatility comes with extra consequences: expensive, slower

³⁴ micro-probing: inserting metallic probes, of sub-micron dimensions, to an open silicon die, to bridge faulty connections or to inject test signals

write speeds and a limited number of write cycles. They are volatile because design requests didn't require them to be non-volatile, not because they were required to be volatile. We will soon see that fully volatile solid-state memory is unfeasible and very expensive.

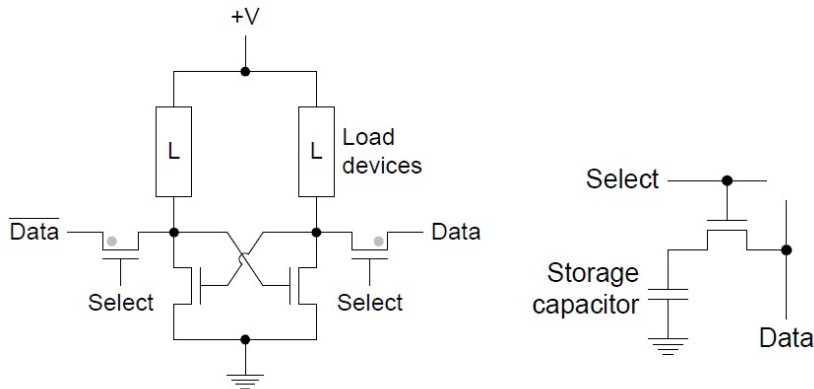


Figure 2.11 - SRAM and DRAM memory cells

Both static RAM (a bistable holding a bit of data) and dynamic RAM (storage capacitor holding a bit of data) consist of MOSFET transistors. They operate with voltages of 5V or lower. What will happen if SRAM or DRAM cell stay in a 0 or 1 state for a long time? It is explained in 2.6.2. A low 5V voltage at a MOSFET's gate for a long time will cause a slow and gradual accumulation of electron charge trapped inside the SiO_2 , with only a few fA (10^{-15}A) of **hot carrier** current (a few electrons out of billions will always be "hot" enough, even at 5V). This will slightly change the threshold voltage V_T of the transistor.

The consequence of a higher V_T is that it will take a higher voltage at the gate (U_{GS}) to make it conduct the same current I_D . Higher U_{GS} means more charge $+Q_G$ at the gate is required. Higher $+Q_G$ required means that we need a stronger and/or longer gate current I_G pulse. If the pulse to turn on the gate remains the same, the **propagation delay**³⁵ will inevitably increase.

A measurement of a drain voltage U_D (relative to GND) pulse when turning on will give information about the amount of hot carrier charge $-Q_H$ trapped on SiO_2 around its gate! Slower rising U_D (or more precisely, a higher propagation delay)- means higher Q_H , meaning the cell has stored a certain bit (1 or 0, depending on cell configuration) for a very long time.

³⁵ propagation delay time (t_p): a time between rise/fall of an input signal and a consequent rise/fall of output signal. In this case, a pulse (current or voltage) on a MOSFET gate is the input signal, and drain voltage is the output signal. Not to be confused with **rise/fall time** (t_R/t_F) which is a time needed for a signal (input or output) to rise from 10% till 90% of its maximum or to **fall** from 90% till 10%.

Individual MOSFET drains are probably not accessible for U_D measurement. However, the power supply current is always accessible (this is how an attack on a Smartcard by **power analysis** is done). If Q_H is high, the power supply current pulse will be higher/longer when turning the MOSFET on.

Here is a simple procedure for how to read a byte from SRAM, removed from the original system, after power off (for minutes or even hours), under the condition that a constant byte had been stored there for a long time.

1. Connect a $10\text{-}20\Omega$ resistor to measure the SRAM supply current.
2. Power up the SRAM chip.
3. Set the address bus input (A0-A15) to the address of the byte to be read.
4. Set test pattern byte to TEST:=0x01.
5. Write 0x00 to the data bus and execute **SRAM write**.
6. Write TEST byte to the data bus and execute **SRAM write**.
7. Record the supply current pulse on a digital oscilloscope.
8. If TEST=0x80, STOP.
9. Shift TEST byte one bit left.
10. Jump to step 5.)

The procedure can run in an infinite loop - at 8.) set TEST:=0x01 and jump to 5.). Then even a plain analogue oscilloscope (without analogue storage memory) can be used if properly triggered. Power supply current pulses will reveal the bits of the byte which are stored there for a long time.

Another, even simpler, variant of this attack can be mounted by measuring power supply current or data bus voltage at the **SRAM read** operation of different byte patterns, **not write**.

Once again, let's be reminded (already mentioned at 2.6.2) that new high-tech RAMs (especially DRAMs, which are specifically targeted for higher and higher data storage densities) are **more susceptible** to these side-effects. Smaller widths and complicated 3-D structures -> higher electric field spikes -> more hot carriers, and since all of this happens on a smaller surface -> less capacitance-> more V_T shift.

Depending on memory IC die technology, most of the trapped hot-electron charge will recover by itself (will not cause the IC failure after 20-30 years, like on (E)EPROMs). The approximate time required for recovery of an average SRAM device at different temperatures is listed in Table 2.2 based on [13]. The criterion was the recovery of **read access time**³⁶. Please note that it may imply that heating an SRAM chip to 75°C for a few hours is a good and very practical method to zeroise it! - Worth testing, along with accelerating the burn-in by heating the SRAM while powered on.

³⁶ read access time - after setting the address bus bits, a time between asserting of RD (initiate memory read request) pin and stabilisation of valid bits at data bus output.

0°C	20°C	50°C	75°C
<i>a few years</i>	<i>a few months</i>	<i>a few days</i>	<i>a few hours</i>

Table 2.2 - Approximate time needed to fully recover SRAM read access time

Many different methods have been proposed by authors, leading in some cases, to quite opposite conclusions. Further research and experiments in this area are definitely required. Before we continue to a less sophisticated, but more reliable variant of this attack, let's explain one possible source for confusion. In section 2.1, I said that defences against attacks that require Eve to get Alice's electronic device in her hands will not be discussed, because Alice and Bob will have to do their best to keep their devices safe and secure. What I meant is that Alice and **Bob can't rely on specially designed secure hardware** (like e.g. credit cards) expected to hold their secrets, even when snatched by Trudy or Mallory. We will see later in chapter 4, that unlike the other DIY devices designed for Alice and Bob (that can be kept in their secure vaults most of the time, and securely destroyed if required), the tamper-evident paper mailbox can be expected to get in the hands of Eve easily, perhaps with a little help from Mallory, **without any need to get Trudy involved**. The Box will be designed with ubiquitous, general-purpose components. Eve may get the chance to try to extract the data from the SRAM memory of the ATmega328P, a common general-purpose MCU.

This is why data-retention countermeasures on general-purpose components are very important for Alice and Bob. They can always build a good blacksmith's furnace to securely destroy every digital bit on their discarded electronic devices at 1500°C, however, they will have to take special care regarding the devices that Eve can be expected to grab at lower temperatures.

Back in 1996, I got my HP48-GX programmable calculator: an electrical engineering student's tool of trade of that time. It still works today and I still use it. A nice handheld, 128KB RAM, 1-bit colour LCD, RS232, and IR port, expandable with two extra plug-in RAM cards. Rechargeable batteries were still considered a luxury back then, the same as flash or EEPROM memory. UV EPROMs were cheap but too unwieldy to fit in a handheld device of this size. So, it held up to mighty 4MB (using **bank-switched**³⁷ RAM expansion) of RAM, where user data and all sorts of extra installed programs were stored, as long as you remembered to replace 3x AAA alkaline batteries (and a CR1616 for each RAM expansion card) a reasonable time after a "low battery" warning. As a 2nd year student, I could understand most of its features, but the screen shown in Figure 2.12. still seemed like a kind of black magic to me, especially after establishing it worked!

37 A certain CPU can directly address only as much RAM (and ROM) as limited by the width of its address bus. If the address bus is 20-bits wide (CPU pins A0-A19), it can directly address $2^{20}=1024\text{KB}=1\text{MB}$ of memory (e.g. 512KB ROM, 512KB RAM). More than this is possible with **bank-switching**. This is achieved by asserting a "chip select" input of one 1MB RAM unit (or **bank**) and holding it active by a bistable (e.g. 74HC273), latched by an I/O request from a CPU. Since an I/O decoder is usually already there, the 74HC273 is the only extra hardware required to fit up to 8 more 1MB banks. Each bank-switching will also consume extra CPU cycles - as many as required to select and write-access an I/O unit.

An algorithm (located in ROM) could at least partially recover your RAM if you screwed up something with proper and timely replacement of batteries. Saved documents and programs that hadn't changed for a **longer period** (unlike temporary data!) could be recovered using procedures explained in this section without any need to bend the laws of physics in our low-level 3D realm.



Figure 2.12 - HP48-GX "black magic" screenshot

2.6.4 • Cold boot attacks

The last state of SRAM before a power-down can be fully recovered even without the need to keep data constant for a long time! According to [15], some newer SRAM chips exhibit dangerously high retention time even at room temperature. The retention time of the very common 32KB SRAM 62256 is shown in the following table. The physics behind this phenomenon are the same as those described in the previous section 2.6.3, but the methods of attack are different.

	-30°C	-20°C	-10°C	0°C	10°C	20°C
<i>VDD grounded</i>	10s	3s	1s	0.5s	0.3s	0.1s
<i>VDD disconnected</i>	30s	10s	2s	1s	0.6s	0.3s

Table 2.3 - The data retention time of 62256 SRAM

If V_{DD} is simply disconnected, the MOSFET gates discharge through MOSFET sources only, while if V_{DD} is grounded, gates capacitances discharge through MOSFET drains as well, so the discharge will be 2-3x faster.

The retention time increases approximately by the factor of 2-4x for every 10°C decrease in temperature. What is the reason? The conductivity of silicon doesn't change that much. On a lightly doped n-type Si (with $N_D=10^{14}/\text{cm}^3$ atoms of donors) conductivity will decrease by the factor of less than 10 when cooled from 20°C to -30°C. Conductivity of a moderately doped n-type Si (with $N_D=10^{16}/\text{cm}^3$ atoms of donors) will have only a slight decrease when cooled in this range.

For gate capacitance to discharge, insulated gate electrons need to **leak through SiO₂** to source (and also drain if V_{DD} is grounded), so **Si conductivity doesn't affect it too much**. At $U_{GS}=5V$, statistically few electrons have enough energy to penetrate through SiO₂ insulation (see 2.6.2), so the discharge is slow, and becomes even slower as U_{GS} decreases. At lower temperature, these electrons are even fewer, which slows the discharge even further.

Here is another way to analyse this: a gate capacitance C_{GS} of an SRAM MOSFET is in the order of 1pF. Its average gate resistance R_{GS} at room temperature is around 100GΩ. This leads to a gate self-discharge time constant of $\tau=R_{GS}C_{GS}=0.1s$, which is consistent with measured results. R_{GS} is highly non-linear, difficult to directly measure, a function of the gate voltage and thermodynamic temperature, i.e. $R_{GS}=R_{GS}(U_{GS},T)$. The R_{GS} increases rapidly as U_{GS} and/or T decrease and “hot” carriers become “colder”, for both reasons.

This is a reason why an SRAM IC can retain the memory while powered down for 30 seconds at -30°C and still correctly operate at this temperature when powered up (depending on a variant). A temperature this low doesn't affect the conductivity of Si much but does affect that very small gate leakage current through SiO₂.

One way to execute a cold-boot attack is to cool the SRAM IC down to at least -20°C while the system is in normal operation. Then turn off the power, remove the SRAM IC from its socket and quickly (within a few seconds) insert it in a programmer, power up, and read. Another, more practical method is to prepare a boot-startup program in EPROM that will first dump the whole SRAM contents (e.g. through a serial port) upon a hard reset, and then jump to original firmware for running regular tasks (e.g. encrypting/decrypting with RSA method). Then start the system and let it run normally. When there is a possibility that SRAM contains a private key to be stolen, first cool down the SRAM (using e.g. a deep-freeze spray) to -20°C, and then execute a cold reset (turn the power off and on after a few seconds). SRAM contents will be retained for a few seconds (will keep the private keys). After a restart, a routine from EPROM will dump the SRAM. The private keys can then be extracted.

This kind of attack can be particularly effective against a multi-user system that may have multiple users logged in (Alice, Bob, and others from their organisation). Trudy may physically approach the server (with an authorisation level of **a janitor**), for regular housekeeping. Please note a very common physical security problem here: **low level** employees (janitors and cleaners) need security clearance **to access all the premises** if you want to maintain basic hygiene. If the security-critical data is carefully stored in RAM only, snatching the whole server computer won't be of much use and will make it very difficult for Trudy to sneak out unnoticed with a cumbersome box in her backpack. Cycling the power with frozen RAM may pass off as a short power failure (or OS crash), giving enough time for Trudy to finish swiping the floor and walk away with all private keys on a USB pen drive in her pocket.

2.6.5 • What can we do DIY?

Now it's time to summarise. The digital memory technology available to Alice and Bob (and everybody else as well) suffers from many data-retention pitfalls. Rectifying them on component or device level is either infeasible or very expensive. Alice and Bob can still do a lot on a DIY budget, so let's review their options:

1. **Reprogram the HDD or SSD controllers** to enable the raw sector-level access commands when needed - for zeroisation. This may require a lot of assembler coding, but the project is not expensive and is mostly about firmware programming. It is absolutely performable on a DIY budget. For what Alice and Bob would gain, it is definitely worth the effort.
2. **Build/buy a good high-temperature furnace** capable of reaching 1500°C. Crude and not very practical, but it at least ensures 100% zeroization with relatively simple means.
3. **Better garbage disposal procedures** - when there is no better option than a dumpster, try to at least not to use one in your neighbourhood. Keep in mind that paper shredders are not 100% effective.
4. **Use UV-EPROM, rather than EEPROM or flash if possible.** It is easier to zeroise, besides all other aforementioned security-related advantages. Keep in mind that faulty and no longer functional ICs can almost always still be read, often with very cheap equipment.
5. **SRAM is better than DRAM in all security aspects.** DRAM requires constant high-frequency refreshing in operation which means more TEMPEST radiation. SRAM is simpler to interface and work with. DRAM is used for high-density, large memories (unlike SRAM, only one transistor per bit is needed) that Alice and Bob don't need at all. SRAM needs less power to keep data. It is capable of operating in power-saving modes. A constant demand to increase the density of DRAM leads to very small memory cells, which increases electric fields (and also makes them less homogenous, with much higher spikes). A high electric field means more residual data retention. Furthermore, so-called bit-flipping protection (see below) works better on SRAM than DRAM.
6. **Build your own RAM recovery attack device** for executing cold-boot attacks and SRAM time measurements as described before (propagation delay, rise/fall time, read/write access time, etc.) to recover remanence data. This device will enable you to test your hardware for remanence, and also zeroise it, now having measurable feedback.
7. **Program a bit-flipping routine.** This is cheap, easy, and very effective. For example, invert all bits in all bytes containing security-critical information in regular time intervals, like every second. If each bit spends an equal amount of time in states 0 and 1, the hot-carrier remanence (also called the burn-in effect) will equally affect all bits, so it won't be possible to recover the bits using the methods from 2.6.3. Please note

that this procedure is not effective against a cold-boot attack.

8. **Introduce temperature monitoring** to activate a zeroisation procedure in case freezing or heat are detected. If the circuit is designed to normally operate at temperatures e.g. down to -20°C (on component level, and also on circuit level), then the zeroisation can be triggered at -10°C or even a higher temperature which may be considered abnormal, depending on whether the device is meant to operate in a lab at room temperature, or in field conditions, hot and cold. This will counter cold-boot attacks.
9. **Build your own SD card copy device (SD copiers).** I will describe how to build this in detail in chapter 5. SD cards are highly insecure, but with these copiers, they can still be used in highly secure crypto-systems!

Besides avoiding using an insecure PC to copy security-critical stuff, copiers will enable Alice to separate “**clean**³⁸” from “**dirty**” SD cards, and thus prevent leakage of information. Besides their specially designed secure hardware, Alice and Bob will also still have to use the usual insecure hardware, like PCs or smartphones for actions like sending previously encrypted e-mails, or VOIP communications. These devices will provide a secure link between the “clean” and “dirty” world. More details will follow.

10. **Zeroise SRAM on every reset.** Introducing this simple subroutine to the device’s firmware will help defend against cold-boot attacks. Repeatedly writing 0x00, 0xFF, 0x00, 0xFF,... several times can zeroise an SRAM byte, unless a constant pattern had been stored for a long time (explained in 2.6.3)
11. **Add fully zeroisable analogue memory circuits to your secure device.** Since we established that every type of digital memory technology suffers from some form of retention, and is hence difficult to zeroize, let’s go analogue! Burning a piece of paper or cassette tape, or over-exposing an analogue film tape securely zeroises the data. This way Eve can’t revert everything to cover up her tampering, and Bob will be informed that something went wrong. More details to follow.
12. **Add some tamper-resistant protections.** Trudy from 2.6.4. will have a very hard time trying to freeze the RAM if the server box is protected with an anti-tamper switch or other tamper protection. Trudy will still stand a good chance though because most real-world security-critical hardware doesn’t have any tamper protection. No kind of tamper protection is perfect, but if it merely extends the amount of time that Trudy will need to defeat it, it may be enough. If impersonating a janitor, she will need to have a very good explanation for having to spend 1 or 2 hours inside a server room. We will look at tamper-resistance problems in more detail in chapters 4 and 5.

³⁸ clean SD card: a brand new, still unused SD card, or one that has never been inserted to any insecure device. May contain security-critical data, but it won’t leak any of it, as long as properly used, in highly-secure hardware only.

13. Use TTL bipolar SRAM ICs instead of the usual CMOS. Retro low-tech may step in again! Bipolar transistor flip-flops can't have several seconds of memory remanence like MOSFETs, because of much lower internal resistances (measured in $k\Omega$ range, rather than $G\Omega$ in MOSFETs). TTL SRAMs had been used a long time ago (in the 1960s) and were quickly replaced by NMOS and then more advanced CMOS technology, mainly because of a higher level of integration and lower power consumption. Furthermore, it is very impractical to implement DRAM with TTL bipolar technology. It is difficult to find a TTL SRAM with more than 1KB of memory, but this is perhaps enough to store highly critical data only.

Many attack methods that may seem expensive, too high-tech, or even "black magic" are not that sophisticated at all, and hence available to even shoestring budget spies. The mathematics and physics behind them are also relatively simple.

Now we know the threats that Alice and Bob will have to defend against, it is time to proceed with designing the electronics that will make it possible!

Emma is not a teenage vixen grabbing Eileen's boyfriend at short notice. E&E should suggest "Eve" and/or "England". They were two young women working in a listening station in a remote village in the north of England during WWII. Their job was to intercept German radio messages, radio-location, taping, archiving, and sending their captures to Bletchley Park for decryption. They would also sort messages according to telegraph operators (identified by audible "hand signatures").

Eileen had a boyfriend in real life, but he was in the navy, and she didn't hear from him for a while. Dashes and dots and unique paces of keying were almost all they could "see" relating to the ongoing war.

One day, while taping a transmission from one German telegrapher (they gave him a codename "Kestrel-25" a month before), Eileen told Emma "Can you picture this guy? His hands must be very gentle..."

The following day, as Eileen arrived a little late, Emma had her headphones on and Kestrel had already started his cat-paw gentle keying. After a millisecond of shock, Eileen suggested renaming him "Tomcat-43".

Chapter 3 • Random Number Generators

3.1 • A good RNG as a necessary link in the security chain

Designing good crypto-systems for Alice and Bob must start with a good TRNG. Random numbers can be used for one-time pads, generating prime numbers for RSA (PGP program does it), one-time AES session keys, printer or monitor anti-TEMPEST scrambling (shuffling the order of scan lines or printed dots), and many other security-critical processes. You simply can't do good crypto without a first-rate TRNG. In 2.2.2 we analysed some methods of attack that Eve (eavesdropping on RNG or re-generating the sequence), Mallory (tampering with RNG or jamming it), and even Trudy (tampering with the delivery of one-time pads) can mount against an RNG. Now let's see how Alice and Bob can defend themselves. First, they need to know how to test and evaluate an RNG for potential use in cryptography.

3.1.1 • Defining specs for a good RNG for use in a crypto system

Many RNGs can be bought as commercial products, but few are reliable for use in crypto systems. Let's first see how Alice can check if a sequence of numbers is random or not. The NIST¹ standard [16] defines a series of 15 mathematical-statistical tests that can be done to evaluate a sequence of random numbers. If it doesn't pass the test, the RNG is bad. Be careful, because on the other hand, if the sequence passes the test, it still doesn't mean the RNG is truly random and of a quality to be used in cryptography!

Passing the NIST test is one requisite (number 0). The others are:

1. The sequence must be generated from a true random process. Any good PRNG will also pass every NIST test before the sequence starts to repeat.
2. All variables and signals must be measurable and observable. Otherwise, Alice can't trust the RNG because then she can't verify the 1st condition. Perhaps it has been tampered with, or it is simply a black box...
3. Designed in a way that Alice can assemble it using a DIY method with easily obtainable ubiquitous components. Special purpose components are easy to trace by Eve and interfere with by Mallory. Outsourcing production means having to trust that 3rd party.
4. Sufficiently fast output, in order of 1Mbit/s. Even the kbit/s range may be acceptable, depending on the application. Slow random processes, like rolling the dice, may have good randomness but are not very practical at cca. 1bit/s.
5. Not connecting to any insecure electronic device, and not networking in any way. This will prevent eavesdropping and tampering.
6. Installed in a well-shielded aluminium box, powered by batteries. This will prevent TEMPEST eavesdropping and jamming.

1 NIST - the US "National Institute of Standards and Technologies". Similar to German **DIN** "Deutsches Institut für Normung" or Russian **ГОСТ** "Государственный Стандарт". Among other technical standards, they also define standards for cryptography.

3.1.2 • NIST testing

According to [16], the 15 tests are defined. Detailed exhaustive testing of a prototype TRNG is necessary before deciding whether to use it for cryptography. The following are 15 NIST tests:

1. Frequency mono bit test

This is simply counting binary ones and zeroes and comparing. An about equal number of ones and zeros is the basic property of any random sequence.

2. Frequency test within a block

The same as 1, but the whole sequence (of n bits) is divided into blocks of M bits, and processed within blocks and then among blocks.

3. The Runs test

The total number of “runs” is counted for a sequence of n bits. A “run” is an uninterrupted sequence of ones or zeroes.

4. Longest run of ones in a block test

Similar to 3, the longest runs of ones in blocks of M bits are counted and compared.

5. The binary matrix rank test

Binary matrices are created from random bits and their ranks are calculated. A too low rank means some rows can be expressed as linear combinations of other matrix rows, and then the sequence is not random.

6. The Discrete Fourier transform test

An FFT transform is done on a sequence of n bits, then the frequency graph is checked for excessive peaks. A good random sequence doesn't have any peaks.

7. The non-overlapping template matching test

Different “templates” of predefined fixed words (usually $m=9$ or 10 bits) are counted in a stream of n bits.

8. The overlapping template matching test

The same as 7, but with a different bitwise shift in a sequence after a match.

9. Maurer's universal statistical test

An attempt to “zip” a file, using a standard procedure (similar to WinZip or WinRar) to uncorrelate the data. The test fails if it is possible to zip a file (which means to express some parts of a file using other parts).

10. The linear complexity test

Based on a shifting register, trying to produce linear combinations between sequences of bits, similar to 5.)

11. The serial test

This calculates the frequencies of occurrence of all possible m-bit words.

12. The approximate entropy test

Like 11, but now the two bit-pattern lengths, m, and m+1 are taken into account.

13. The cumulative sums test

This is a test of random “walks” around the zero point based on random numbers sums, and checking the maximum deviation from zero. A sum of a long sequence of good random numbers in a range $[-R, +R]$ must always remain close to zero.

14. The random excursions test

Like 13., but now all deviations from position zero are checked for frequencies of occurrence.

15. The random excursions variance test

Similar to 14, but now statistical variance is evaluated.

These tests are very precise and rigid (which is needed to guarantee good security for a crypto-system), may be confusing at first glance, but in essence, they are based on the following intuitive and easy to understand principles of randomness:

1. A truly random bit stream has an equal number of ones and zeros.
2. A truly random bit stream has equal frequencies of occurrences of different bit patterns.
3. No part of the bit stream is mathematically correlated to any other part of the bit stream.

Before doing the detailed NIST testing, the following simple tests can give you a rough estimate:

- 1.) Generating a **histogram** using a program like WinHex. All 8-bit patterns (from 0x00

till 0xFF, on the x-axis) are counted and their numbers of occurrences are plotted on the y-axis. They should all be about equal. See [17] for examples.

2.) Try to compress a file containing random numbers, using WinZip or similar software. If the randomness is good, the “compressed” file ought to be even larger than the original.

3.) Try to simulate Buffon’s problem² using random numbers under test. If the randomness is good, the precision of a calculated approximation of π must constantly increase.

4.) Use the random numbers to generate a bitmapped image, then visually observe the image for recurring patterns. If they are noticed, then this is PRNG, not TRNG, as the sequence repeats.

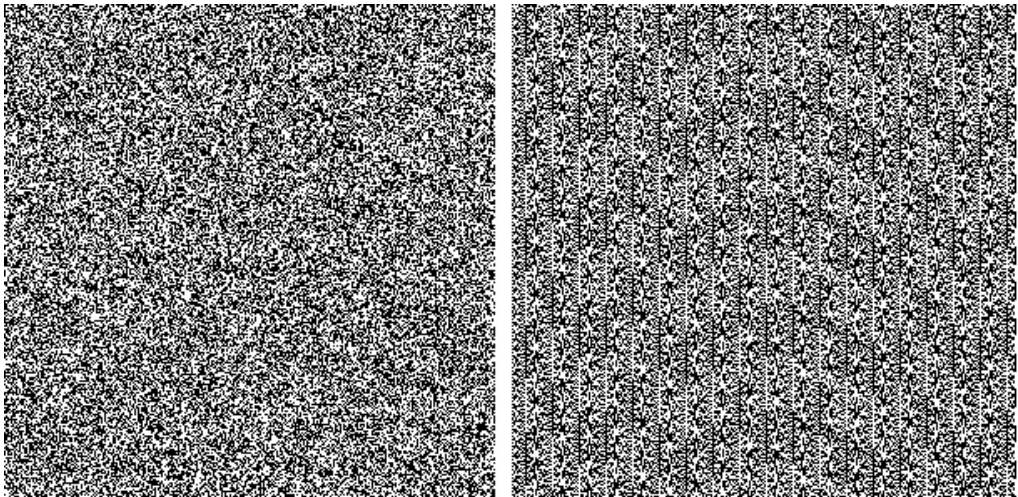


Figure 3.1 - Humans, like many other animals, are very good at quickly noticing **patterns**. TRNG and PRNG outputs are very easy to distinguish at a glance.

3.1.3 • Other ways to use NIST RNG tests for security evaluation

Before continuing to talk about the different ways to implement RNGs, let’s see some common applications of random number sequence quality testing. Apart from testing the quality of an RNG output, these tests can also be used to evaluate the performance of many other processes.

1. Output from any good cryptographic encryption function (OTP, RSA, AES, etc) should have good random properties, to pass NIST testing. If every bit pattern at output has **the same probability** of (random) occurrence, it becomes more difficult to crack the encryption, as mentioned in 2.2. Ibn al-Durayhim figured out how to crack the

² The probability that a needle of length a will fall on one of the parallel lines, drawn with distance d between the lines can be calculated as $p = (2a)/(nd)$. An approximation of n can consequently be calculated by dropping a needle a number of times, or by mathematically simulating the drops using calculations from pre-generated random numbers.

mono-alphabetic cipher in the 14th century just because Arabic (the same as any other language) has a **different probability** of the occurrence of particular letters and letter groups in normal plaintext. NIST testing of output can be used to evaluate any encryption method.

2. Similar to 1, NIST testing can be used to evaluate cryptographic hash functions (e.g. SHA-1). Changing a single bit in a hash function input must cause a large change in the output, and any output pattern must have the same probability of occurrence, otherwise, the hash function input can be partially deduced from its output. Hash functions were purposely invented to make inversion impossible (or at least very difficult).

3.2 • Types of RNGs available today and possible problems

Many different RNGs have been designed and exploit diverse random natural random phenomena (or simulate them), implemented in different ways, on various hardware, ranging from highly integrated one-chips to many discreet components on a regular PCB. After an extensive search, I found almost none of them fulfils the crucial requirements for use in cryptography (defined in 3.1.1). This is why I decided to design my own TRNG.

3.2.1 • Pseudo-random numbers generators (PRNG)

Although not truly random, they can still be used even in crypto systems, with certain strict limitations. The simplest type of a PRNG is a so-called **linear-congruential generator**, proposed in 1948, according to [7]. It works based on a simple recursive formula:

$$x_{i+1} = (ax_i + c) \bmod m$$

The new pseudo-random non-negative integer number x_{i+1} is calculated based on the previous value x_i . Function “mod” is a remainder of an integer division after the expression in brackets is divided by m . Number m is called a *modulus*. A division with m can yield a remainder in a range of $[0, m-1]$ (limits included), meaning all pseudo-random numbers x_i will be constrained in this range. The initial value x_0 is called a *seed*. With the carefully chosen values a and m , the PRNG will generate m different values before the sequence starts repeating. For example, if $m=24$, $a=6$, $c=5$, $x_0=1$ the sequence will become 1, 11, 23, 23, 23... After only three different values (out of $[0,23]$ range - which is 24 possible values) it will become stuck at 23 and stay there.

For the PRNG to work (to generate m different values before the sequence starts repeating), m must be a **prime number**. Additionally, number a must be a **primitive root** of the number m (this is explained in [7]). Number c can be any non-zero value. This is because if $c=0$, and $x_i=0$, the sequence will get stuck at $x_k=0$, even if a and m are chosen correctly. This generator will produce sequences with good random qualities (tested by NIST or any other method). The sequence will start to repeat after exactly m steps. NIST tests will then signify the deterioration of random qualities. I learned this very well when I programmed my Speccy to simulate the Buffon’s needle drops to calculate the π number. The Spectrum’s

BASIC uses the RND function to generate a pseudo-random number, with $a=75$, $c=74$, and $m=65537$ (all set fixed in the ROM). Two random numbers are needed to simulate every needle drop (one number for the distance between the needle's centre and the line, and one for the needle's angle of rotation). It took a while for my Speccy to simulate more than 32000 needle drops. The value of n was steadily improving, before starting to deteriorate. This way, I learned the difference between true-random and pseudo-random very well! Testing of true randomness seemed like black magic to me at first, but this very simple test, possible to be successfully executed on a ZX Spectrum, showed the difference in a scientifically acceptable manner!

For any kind of use in cryptography, the m value must be in the order of at least 10^9 . Alice and Bob must both set the same values of a, c, m , and x_0 . As long as they keep the key numbers (i.e. a, c, m , and x_0) secret and large enough, Eve will have a hard time trying to crack the code. However, she might eventually succeed. Cryptography devices using PRNGs usually work in a way where they periodically reseed the PRNG using a much shorter pre-recorded sequence of true random numbers. One type ("**KZU-x**" used by the Croatian military until recently, now de-classified) used a punched paper tape to store a short TRNG sequence (because paper tape is very easy to destroy).

Values a , c , m , and x_0 at this order of magnitude are very impractical for calculations in real-time. However, a combination of several linear-congruential generators can have a much higher period with numbers that even a Speccy could handle without difficulties. The following is the **Wichmann-Hill generator** [7], with a period in the order of 10^{12} .

$$\begin{aligned}x_{i+1} &= 171x_i \bmod 30269 \\y_{i+1} &= 172y_i \bmod 30307 \\z_{i+1} &= 170z_i \bmod 30323 \\u_i &= x_i + y_i + z_i\end{aligned}$$

Initial values (seeds) x_0 , y_0 , and z_0 may be used as crypto keys, reseeded with pre-recorded true random numbers from time to time.

Non-linear congruential generators have also been researched. A **Blum-Blum-Shub** generator [7] is based on the following simple non-linear formula:

$$x_{i+1} = x_i^2 \bmod m$$

According to their recommendations, m is chosen as $m=p_1p_2$. Numbers p_1 and p_2 are two large and different prime numbers, each giving a remainder of 3 when divided by 4. Under these conditions, the sequences generated show very good random properties, up to the cryptography level. Now is a good time to note that **prime** numbers are the second most important type of numbers used in cryptography, with **random** numbers being the first. Because a CPU is already there, PRNGs don't require any special extra hardware to be implemented within a crypto system. This fulfils the number 3 requirement completely, listed in section 3.1.1. Tuning a PRNG to pass the NIST testing is relatively easy. This will fulfil the number 0. All PRNG variables are easy to monitor, so number 2 is also met. Output

in the kbit/s range can be achieved, thus partially fulfilling requirement number 4. Even 5 and 6 are easily achieved because PRNGs don't operate with any kind of weak analogue signals that are easy to interfere with. Only crucial condition number 1 can not be met using a PRNG. This is not good. Although they can reach a relatively high level of security, their use in secure crypto systems is not recommended. The truth is that there have been much better options available to Alice and Bob, essentially since the early 1980s. Let's just review a few, in fact much inferior RNG variants for cryptography first.

3.2.2 • Highly integrated TRNGs

Many variants, based on different random processes have been successfully designed in the recent past. Article [18] is about a highly integrated TRNG based on a thermal noise affecting an oscillator's **jitter**³, micro-sized for integration on any Smartcard silicon die. Atmel's 32-bit MCU Cortex-M3 SAM3X-A features an integrated TRNG based on two oscillators with slightly varying frequencies. There are many other types, have been around for a while, and many of them are easily obtainable.

Alice and Bob could use one of them, why not? Some of them have been around long enough that their patents have expired. The main problem is that Alice and Bob probably can't produce microelectronic circuits themselves which are too expensive and complicated for their makeshift basement labs. If not, they will have to **trust** somebody, actually many other persons, from the designers of the TRNG IC (to trust that their specs are as good as they claim), manufacturers (to make everything up to the specs in a cheap Chinese factory and not to be bribed to install hardware Trojans) and also delivery (not to allow Mallory to plant her own rigged ICs). Spies who trust a lot of people simply don't celebrate many birthdays. Practicing the world's second oldest profession, they have had enough time to verify this simple fact.

All problems of highly integrated devices were already discussed in 1.1.3 and 2.5.2. They are simply not good for cryptography and can't be trusted. All highly integrated TRNGs are also black-boxes, without any exceptions.

3.2.3 • Black-box TRNGs

There are also many other TRNGs, some based on quantum effects and expensive hardware, with internal specifications that are kept secret. They can be purchased legally, but it is illegal to open them to look inside. A good quantum physicist and expensive lab equipment would be required to analyse them. They suffer the same problems as 3.2.2. On top of this, they are too expensive for Alice and Bob. They need something cheap they can assemble on a DIY budget where it is straightforward to check every signal, in order to be trusted. This brings us to my first successful project completed with Elektor [17].

3 jitter: a deviation from a perfectly constant time period of an oscillator

3.3 • Elektor TRNG solves some problems, but...

Avalanche⁴ noise on some types of 12V Zener diodes (e.g. BZX85C12) has a sufficiently high amplitude, and good frequency spectrum (up to 10MHz). It is fast and random enough to generate good random numbers, up to 1Mbit/s. All components required are general-purpose and easily obtainable for Alice and Bob. Avalanche noise signal needs to be processed, amplified and filtered before the MCU can extract a stream of random bits.

⁴ Avalanche noise is a type of noise which occurs on a reversely polarised lightly-doped Zener diode, during a stable non-destructive breakdown, at voltages higher than 6V.

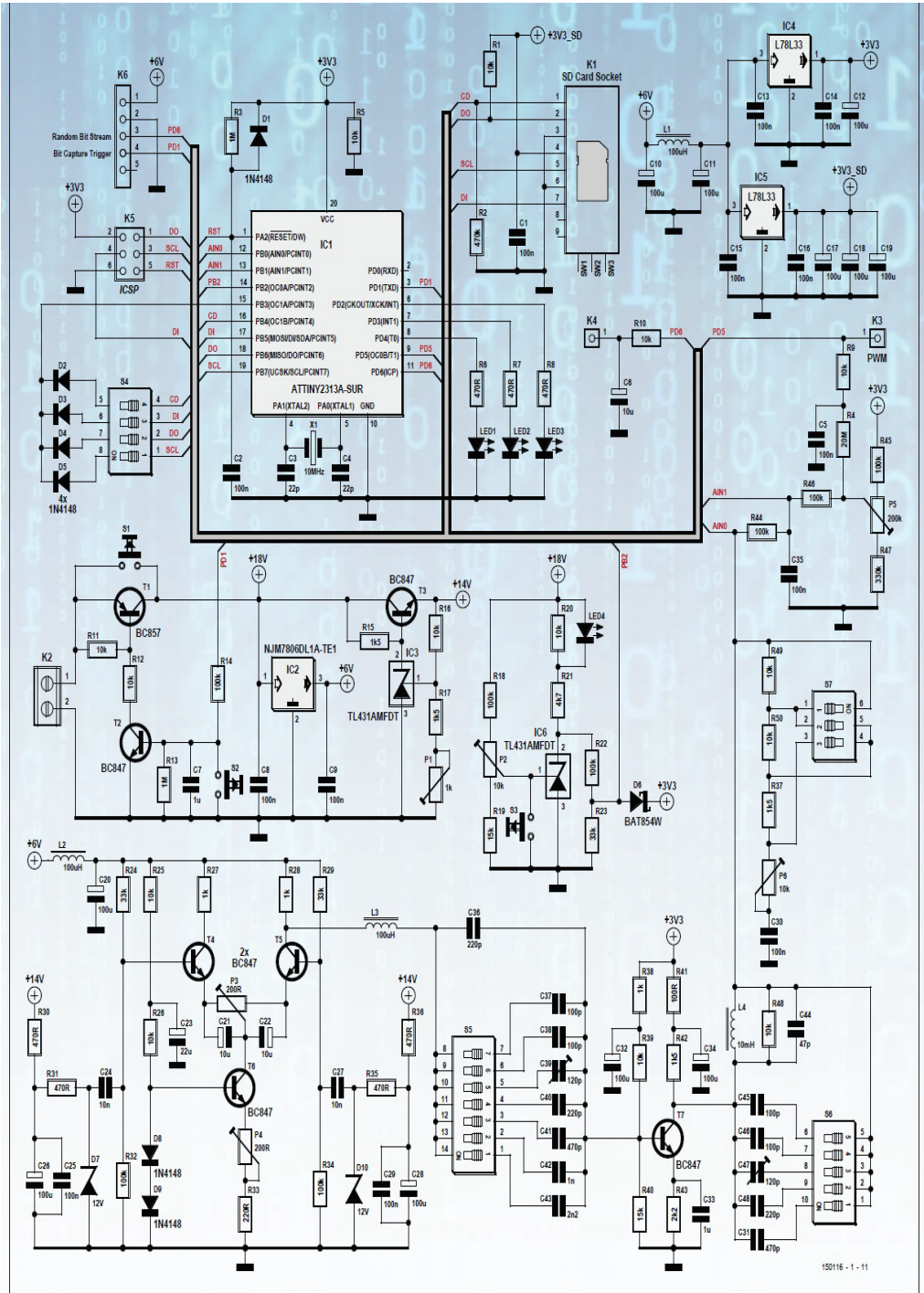


Figure 3.2 - Elektor TRNG schematics

Noise on one diode is a random process but is also affected by various deterministic influences. This is why two 12V Zeners (D7 and D10) are used and amplified by a differential amplifier (T4, T5, and T6). The basic textbook property of a differential amplifier is to amplify a difference signal and reject a **common-mode signal**. A **difference signal** is useful here, because it is a difference between two random signals, and is highly random. Common-mode influence is about the same on both diodes (e.g. environmental influence, like temperature and everything else) and it is cancelled out. The same textbook says that CMRR⁵ is amplified by increasing the dynamic resistance common to T4 and T5 emitters. This is why T6 is used, wired as a current source, with very high output resistance.

This is the first step to filter and improve the useful random signal. An analogue corrective filter built around T7 will emphasise and suppress certain spans in a 25kHz-4MHz range affecting the quality of random sequences. Careful tuning of this filter must be done to ensure all 8-bit patterns achieve approximately the same frequencies of occurrence in a histogram, which is essential for cryptography.

PWM⁶ square waves coming from the MCU on pin PD5-K3 are used to automatically compensate the DC drift, before the built-in analogue comparator on the MCU (its AIN0 and AIN1 inputs) can extract the random bit-stream. DC drift compensation is needed to balance the 0/1 ratio, which is also very important for use in cryptography.

The MCU will extract the random bits, and record the bit-stream on the SD card, taking care to arrange them in a properly readable FAT32 file. The bit-stream analyzer (BSA) is an auxiliary instrument used to properly adjust the filters around T7. All the other details are explained in my article [17].

Let's now see how the Elektor TRNG stands regarding the criteria stated in section 3.1.1. Carefully tuning the filters will make the TRNG pass the NIST tests, so number 0 is met. A difference between two avalanche noises is a highly random signal, which fulfils number 1. All signals can be checked, measured, and adjusted, so number 2 is OK, except for possible tampering within the ATtiny2313 itself (Mallory planting a pre-rigged version in Alice's mail). However, even this can be improved and we will get back to this shortly. All components are general-purpose and are easily obtainable: Alice won't draw any attention by ordering them. Since everything is open-hardware and open-source software, Alice and Bob can improve anything they want. They need to know the basics of electronics to assemble and test everything in their makeshift basement lab, without any need to delve deeper into advanced quantum physics. Number 3 checked. Output bit-stream ticks at 0.5-1.0Mbit/s, which will create gigabytes of random numbers in a few hours, enough for years of secret communications - number 4 is good. The secret bit-stream is saved directly to the SD card, meaning there is no need for a connection to any network or insecure device. This meets number 5. Since it can be self-contained (especially if powered by batteries) and kept in a well-sealed aluminium Faraday cage during operation, number 6

5 CMRR: common mode rejection ratio is a A_d/A_{cm} ratio, i.e. a ratio between differential and common-mode amplification - the most important property of a differential amplifier.

6 PWM: pulse-width modulation is achieved by changing the on/off (or so called mark/space) ratio of a constant-frequency square-wave. It effectively changes its DC level.

is also OK!

Now let's analyse the potential for attack on this type of RNG, according to 2.2.2. Eavesdropping on this RNG (especially if closed inside aluminium box and powered by batteries) is extremely difficult, so condition number 1 is met. Regenerating a sequence generated from electronic noise is not possible, which checks number 2. If Alice assembles the RNG herself, the only possibility to tamper is by tampering with MCU. If this can be avoided, number 3 is met (the same as number 2 in section 3.1.1). Jamming the RNG contained in a good aluminium box is next to impossible, so number 4 is checked. Using this RNG, Alice and Bob can generate secure random numbers for only themselves. It will be therefore difficult to tamper with the delivery of one-time pads, which finally fulfils number 5.

All problems solved? Well, almost. Post-processing (e.g. XOR-ing two long sequences, as mentioned in [17]) can further increase the quality of random streams. We can replace the ATTiny2313 with a simple Z80 system (CPU, SRAM, UV EPROM, and analogue comparator as separated ICs), to mitigate the possibility of hardware-Trojan tampering. We will see about this in section 5.3. Please remember that TRNG (the same goes for many other crypto-devices as well) doesn't need to be pocket-size, especially if intended to be kept in a secure vault and used occasionally.

The problem yet to be solved is copying this SD card (Alice and Bob each need to have a copy) securely, and copying this SD card to a medium more acceptable from a security standpoint, such as a cassette tape. We will explain in detail how to build these two copiers in sections 5.1 and 5.2. Even with these much simpler devices than the TRNG, they will greatly increase overall security.

Now that Alice and Bob can securely make copies, they will sometimes need to send copies to each other when they are unable to physically meet. They can use the tamper-evident box described in chapter 4 for land-mailing, which will detect tampering and alert Bob that the SD card sent by Alice is potentially insecure. Bob can then discard it, and generate another using the TRNG (or ask for another from Alice).

In this chapter, we showed why and how to make a good TRNG. It has been subjected to numerous tests, and the firmware has been improved several times, as you can see on Labs project pages. Our German electronics engineer colleague, Matthias Wolf from Munich decided to construct a variant [19], capable of **displaying a live histogram** on a PC monitor **through USB**. This is more convenient and precise for filter adjustments than a simple BSA. For random sequence recording, USB can be disconnected after the precise filter tuning is completed.

As you can see, new problems have just emerged. We will need to solve them too. Following our already well-defined design philosophy, let's continue with making a simple and secure encryption device!

1950s West Germany, a NATO base:

A Soviet spy managed to successfully infiltrate NATO - a pro-Soviet Polish posing as an ethnic German from the west of Poland escaped from a Russian POW¹ camp after WWII. As the NATO base cryptologists managed to crack encrypted radio transmissions from the East, he could alert them to change their codes in time.

One day, he was in an office with a cryptographer who angrily threw away a paper with a message intercepted from a STASI radio telegrapher.

"What is it?"

"Makes no sense. Some garbage. Commies must have changed the codes..."

He picked this paper up from the trashcan an hour later. Two hours later he decided to blow his cover and defect to NATO.

Why?

1 Prisoner of War

Chapter 4 • Cryptography on paper, computer, and in the real world

Specs for a good crypto device must be first carefully defined. As we have seen, many protection schemes have failed in the past, simply because threats were not properly identified. This means the protections were **not defeated but bypassed**. This is a big difference. This is what happened with the Maginot line at the beginning of WWII - well designed for mostly static trench warfare of WWI but easily **bypassed** by unpredicted modern highly-mobile Blitzkrieg of WWII. Both chapters 1 and 2, in particular section 1.2.2 mentioned many such examples.

4.1 • Why do cryptosystems fail?

The whole of chapter 2 was about how to defeat a crypto-system. There are, of course, many other ways, some of them yet unknown to the world's top security experts. Even if the mathematics of cryptography is **perfect**, electronic real-world implementations are **always much less perfect**, and humans handling electronic devices and secrets in the real world are **always far from perfect**. Let's first review some more historical examples (the anecdote from the beginning of this chapter (allegedly a true event), is also one of them).

4.1.1 • The famous ENIGMA

We know it eventually failed, in a way that its encryption was cracked. Its hardware implementation was OK and the German personnel using it were well educated and disciplined. Even after a few fully working machines were captured by the Allies, they were still unable to quickly crack the encryption. This proves that it was designed, assembled, and implemented following all textbook rules - when the encryption algorithms became known to their adversaries, the real secrets i.e. the keys (rotor settings), remained secret, which kept the communications secret. This basic principle of cryptography was explained in 2.2.2.

Furthermore, the critical operations of encrypting/decrypting messages with ENIGMA machine, and transmitting/receiving ciphertext with a radiotelegraph **were physically separated**, which is crucial for good security (already mentioned several times, concerning different security aspects in chapters 1 and 2). There was no chance of TEMPEST leakage from ENIGMA operating at very low frequencies, especially inside a steel U-boat surrounded by salt water. Eve (a whole team in the Bletchley park lab, with top-class Polish cryptographers and Alan Turing) could not effectively crack the code (at least not until 1943), Mallory could do something by planting a known-plaintext (a so-called **"chosen-plaintext attack"**) and Trudy could be of very little practical use, especially in a submarine. It is very important to note that none of the attack methods mentioned in chapter 2, intended for electronic digital systems could have worked against ENIGMA, for very obvious reasons.

Ciphertext-only attacks (explained in 2.2.1) didn't work. Brute-forcing would take days. German submarines changed keys every day and reading their plaintexts after a shipping

convoy had already been obliterated wasn't particularly useful. **Known-plaintext** attacks were more successful, because this would quickly narrow the search, if a ciphertext contained some usual, frequently used phrases like "Nothing to report", "Everything OK", "Heil Hitler" or "Mission accomplished". **Chosen-plaintext** attacks were even more successful, when possible, by intentionally causing small incidents in towns with exotic names, the already known plaintext (a town's name) would be "injected" into German official reports for a few following days.

Important to note about ENIGMA was that it was a well-designed system, in all aspects, including security procedures, fully exploiting the technology available at the time. What is more important, is that it didn't **push the technology to the limits**! There was no serious leakage of security-critical information from the German side (unlike what happened to NATO during the Cold War with Kim Philby). ENIGMA's encryption using a constellation of 3 to 5 rotary encoders, linearly advanced after each key is pressed (similar to the mechanical odometer on an old car's dashboard - this was its main weakness), isn't mathematically strong and can be brute-forced (using today's computer technology), but it was strong enough against electro-mechanical computers of the 1940s. Occasional procedural errors, like sending easy-to-predict messages were what enabled Eve to crack the messages, but only for that day.

In the past, many cryptosystems have been defeated by bypassing protections, stealing information, planting viruses, bribery, black-mailing, dumpster-diving... etc. Many other effective methods have little to do with crypto-analysis. This means exploiting poor procedures and bad designs (very common nowadays), is often easier than cracking encryption. ENIGMA is one of not so many systems that were defeated using crypto-analysis!

This is why I decided to study the principles behind the design of ENIGMA, to apply this knowledge in designing my crypto devices. Technology advances quickly, but the basic principles remain valid forever. Unnecessarily pushing the new technology (always poorly and/or insufficiently tested for security at that moment) is the most common mistake that even experienced design engineers routinely make.

For everyone, without exception, reading history books is a must.

4.1.1.2 • VENONA affair

Although being the only mathematically proven unbreakable method, even the OTP (one-time pad) can fail if not used properly. The math behind the OTP is extremely simple, but the math required to crack the code is also very simple. Only a few lines of equations are enough (as shown in 2.2) - if two different messages are encrypted with the same key

sequence. Cracking it then becomes as simple as a **dictionary attack**², and much simpler than cracking a password hash, because hash functions (explained in 2.5.3) are designed to give very different outputs for a 1-bit difference in input.

As explained in 2.2, cracking the OTP is simple if Eve intercepts two different ciphertext messages C1 and C2 encrypted with the same random key sequence K. Shifting through C1 and C2 to align them concerning K (or to say in other words, to synchronise the starting points) is a bit more difficult, it is usually an iterative process of shifting one message by one letter and performing the procedure in the following paragraph.

Eve first calculates $C=C1\oplus C2$ and then tries different meaningful words and sentences for P2. A meaningful $P1=C\oplus P2$ will start to emerge, and Eve will advance with subsequent educated-guessed words in P2 yielding meaningful P1 words. This is **simpler than cracking a hash** because changing one bit in the input letter of the XOR operation changes a corresponding bit in its output only. Eve doesn't even need to make word-by-word guesswork, since she will notice even meaningful groups of 3-4 letters as they appear within a word. If a tested dictionary word P2 yields a good dictionary word P1, both words are OK, and she advances to the next P2 word, until the end of the shorter message. Failing to get a correct initial word P1 after testing many words P2 means that C1 and C2 are not well aligned or even not encrypted with the same random key sequence K.

As you can see, the procedure is simple enough to be performed on a very simple computer, even with the pen and paper method for short messages. This is exactly what the VENONA affair was about, cracking the OTP encryption by Eve, because Alice and Bob exhausted their random sequences (the one-time pads), and then started to reuse them because they couldn't create and/or deliver new fresh pads to each other. When designing my crypto-hardware, I had this situation in mind, so did my best to prevent something like this from happening.

VENONA started as a USA-led counter-intelligence project in 1943, directed against their then ally, the USSR. It kept going until the 1980s. Numerous high-ranking US officials were found to be transmitting sensitive secrets to Josef Stalin, motivated by different reasons, as already discussed in 1.2.5. Besides the aforementioned, some people considered the USSR to be an ally of the USA (which was technically true until the end of WWII), so they thought that they were not spying, but helping their ally by keeping them up to date. The Americans managed to crack many OTP encrypted messages in the 1940s, mainly because Russians reused old random numbers several times in that period. Anyway, all crucial information concerning the Manhattan Project (development of the first nuclear bomb) wasn't prevented from being leaked to Russians, enabling them to quickly catch up in the

² dictionary attack: performed by trying all the words from a dictionary (if the language of the plaintext is known). It is a kind of a brute-force attack, although with far less combinations, because the number of all possible combinations of letters (e.g. all 6-letter words of English alphabet, $26^6=309$ million, and many English words are longer than 6 letters) is much higher than a total number of words in English dictionary, which is less than 1 million. Typically used when a hash of a password is known, to extract the password. When all plain dictionary words are tested, it continues with different uppercase and lowercase letters, forward and backwards, putting numbers at various places, etc...

nuclear arms race.

Generating huge amounts of truly random numbers, for every Russian field agent was a big problem in the 1940s. Electronic computers were still in their early infancy. Therefore the technical resources needed to generate and duplicate random numbers were very limited. The generation, duplication and secure distribution of one-time pads with 1940s technology was practically impossible. You will see how I solved each of these problems (you already know about the TRNG), but I needed at least mid-1970s technology.

There is one good engineering mind-training exercise that any design engineer should practice from time to time. After successfully solving a technical problem, try asking yourself how could our comrades **from the past** may have solved the same problem? Make just a little bit of “a time travel”, back to 1940s, 1950s,...,1970s,... Remember the “Back to the Future” sequels? The third Wild West instalment was in line with this - **how-to reach 140+ km/h speed with 1880s technology** because DeLorean ran out of fuel? Pushing a standard steam-powered piston-engine locomotive from that era well over its design limits (and the railway tracks as well: there are many railroad sections in **Europe nowadays** that are considered unsafe even for 100 km/h) is but one (very dangerous) method. Although steam locomotives capable of safely reaching 200+ km/h have been later developed, the standard models available at the time had extremely poor rough-edged airframes (no streamlining). Even if the boiler and/or firebox³ didn’t fail due to extreme pressure and temperature stress (more speed requires higher steam pressure in a boiler and also higher output flow of steam, which requires much higher temperature in a firebox -simple, isn’t it?), perhaps excessive aerodynamic forces (both lift and drag, increasing with **speed squared**) would break or derail it. Operating an external combustion engine manually (not much automation was available back then, maybe a centrifugal speed controller-stabilizer and boiler pressure-relief valves, which **both had to be disabled** for this to work) while **ripping the envelope**⁴ is very difficult. Maybe Marty and Doc could do better with makeshift solid-fuel rocket propulsion akin to a “rich” gunpowder mixture with more charcoal and less saltpeter, so it wouldn’t explode, but only burn very fast? Or even better, slap up a makeshift refinery to distill only a few litres of fuel since they don’t need the DeLorean’s engine to run longer than a few minutes before the low-quality “**moonshine**”⁵ petrol kills it? This kind of failure is far less dangerous for a driver than an explosion from a steam boiler or homemade “rocket engine”. Can you think of another method?

Ask yourself the same question **for the future** e.g. take into account what will become possible when quantum computers become capable of factoring semi-prime numbers higher than $21=3 \times 7$. Maybe you used too high-tech, which is less secure? Or maybe too low, e.g. perhaps your encryption was strong enough for 1950s state of the art but possible

3 A furnace of a solid-fuel steam engine

4 To “**force the envelope**” means to operate a machine **on the edge** if its safe operating limits (“envelope” is a curve -a mathematical function limiting the safe area, of e.g. temperature, pressure and speed). To “**rip the envelope**” means to do it **well beyond** the safe limits. Requires nerves of steel or extreme unawareness or finding someone else to do it for you or simply using remote control.

5 A slang word coined during the 1930s period of prohibition in the USA. Refers to illegally produced alcoholic beverages, done homemade-DIY secretly at night, to avoid detection.

to be easily cracked in the 1980s? I concluded that this kind of mental exercise is always important, no matter what kind of design engineering you are into, and necessary when dealing with secure crypto devices. Regardless of all the errors of the past, many of our colleagues routinely forget to study older technology.

The Russians learned their lesson in the 1940s and consequently improved their methods of handling one-time pads securely. Now it's your turn to figure out the next logical question. If they couldn't do it with 1940s technology, how did they make it with 1950s state-of-art (analogue noise properties well understood, analogue tube amplifiers developed well enough, discreet transistor technology still at an early stage, and no integrated circuits available)?

4.1.3 • Mathematics is perfect - well almost...

Maths is the most perfect natural science of all. It never fails. Good cryptography always starts with good maths, and I like it. It's nice to have something perfect to start with, right? The good thing about maths is the following: If a mathematician-cryptologist finds a reasonable way to crack encryption, then it is proven that the encryption is breakable. The problems start when you try to turn this implication the other way. How can you prove that encryption is unbreakable?

The answer is that the only proven unbreakable encryption is OTP, **because it is easy to prove that it can't be brute-forced**. The key is as long as the plaintext, so any plaintext is possible with any one-time pad key random sequence. It is not possible to find a key, and decrypt the rest of the ciphertext with it, thus proving the key to be correct. For any other method, where a key is always shorter than a plaintext, the key can theoretically be brute-forced, which immediately means the encryption is breakable.

So, you can calculate the number of combinations in the keyspace and thus estimate the strength of encryption. The problems start when math advances (after some time) and a mathematician finds a method to quickly reduce the keyspace search. This way it becomes less brute force. Just remember what Ibn al-Durayhim did in the 14th century. Even the mono-alphabetic cipher had been considered unbreakable until then.

Cracking old algorithms, once considered perfectly secure amplifies the need for researching new, stronger methods. It takes a lot of time, and many experts to properly evaluate them. For example, the SHA-1 hash function was invented after older hash functions (e.g. MD-5) had shown weaknesses. A perfect hash function has no calculable correlation from output to input, and brute-force is the only method to find an input based on output. After a while, weaknesses were found in SHA-1, and then SHA-256 was introduced, which is a standard for the Bitcoin blockchain today. Many even stronger SHAs have followed. Needless to say, if somebody finds a way to quickly reverse the SHA-256, the Bitcoin will devalue faster than the German mark in 1923 or the Yugoslav dinar in 1990. On the other hand, cryptocurrencies with much stronger crypto-protections (like e.g. Z-cash) than original Bitcoin are already in circulation, which is why there are good prospects for the development of blockchain-based monetary systems since paper-cash nowadays is also no longer backed

by gold or anything of stable value. It is always much easier to invent a stronger hash function for authentication (or at least to increase the number of bits of an old one) than to find an effective way to crack it. Linearly increasing the number of bits increases the time required to brute-force the hash exponentially. Finding a mathematical weakness which would enable to quickly crack a new hash without brute-force usually takes a lot of man-hours and is not always effective.

This is why maths is **almost** perfect when it comes to evaluating crypto security. In the general case, maths **can't** tell you **how secure** a certain algorithm is in a given moment at present! It can only give you an estimate, based on the knowledge available now. Sometimes it is useful: some encrypted messages are perfectly OK to be cracked 24 hours later (like coordinating a "wolfpack" of U-boats to attack a shipping convoy in the North Atlantic), or one-time banking transaction session, after the session key expires within 10 minutes. On the other hand, some encrypted messages hold extremely sensitive information for the people involved who would sometimes like to keep the status quo forever.

4.1.4 • Humans are definitely not perfect

Yes, we are far from perfect, even the professionals directly involved in the security business. The Kim Philby affair, mentioned in 1.2.5 reveals only a small fraction of human incompetence to properly handle security. Of course, there is always more.

Until the end of the 19th century, electronic security was practically irrelevant. Somebody, please tell me a story about eavesdropping on wired Morse telegraph, or cracking its encryption, with a significant effect. Until maybe 50 years ago, mostly professionals needed to exercise any kind of electronic security caution. Common folk had practically nothing to do with electronic security. Did we all use Smartcards in the 1970s? How about electronic car keys or remote banking software? What kind of secure wireless RF signal did a common citizen ever need to transmit up until the 1990s (maybe from cordless analogue landline telephones, then unencrypted and very easy to crack)? Did we all have a Smart TV with an integrated camera, in every living room, connected to the internet, always ready to report on our suspicious everyday activities?

A 20th century mathematician, whose name I can't remember (sorry) said that human brains are well developed for hiding from the rain, counting to 10, recognising edible berries, and evading carnivorous predators. He continued by saying that we are simply not designed for thinking in more than 3 dimensions or handling big numbers, especially after tens of thousands of years of persistently tuning our brains and eyes (using state-of-the-art natural **genetic**⁶ neural-network training algorithms, better known as survival-of-the-fittest) to quickly detect a 10-15Hz flicker (a typical snake rattling frequency). He continued by saying that we had made a big mistake by having descended from trees in the first place (this one he had probably "stolen" from Douglas Adams), let alone learning to light a

⁶ One type of advanced control algorithms in automation theory is actually called "**genetic algorithms**". These algorithms mimic the "natural selection" processes related to natural biological survival of genes carrying better properties.

torch or to sharpen a rock. As far as I have seen, my Toxy⁷ does a much better job in most aspects important for survival. He is more effective in nocturnal hunting, equipped with superior built-in night vision, less noisy and more stealthy, better armed - needs neither torch nor spear, faster to detect and escape big predators, except maybe for counting.

Sounds ironic and misanthropic, but my interpretation is that he wanted to say that **learning to use** new technology (e.g. fire or long-range ballistic weapons⁸) is much easier than learning to use it in a fully **responsible and conscious manner**, which takes much longer. Many more people use computers today than 50 years ago, but much fewer are aware of the risks and pitfalls. There is even less interest in learning about it today than 50 years ago. Back then, an average computer user was a highly skilled professional, operating on an expensive machine that he respected and didn't want to break. The Cold War was in the full "heat", so he was certainly more aware of hostile spies/hackers than any computer user today.

I am not saying today's humans are simply stupid. They are simply **much less motivated** to learn about proper security procedures. Security procedures are considered a nuisance by most. Dangers today are less obvious than they were in the past (whether KGB⁹ or CIA spooks, or cave-dwelling predators), and this is not likely to change anytime soon, along with our genetics.

4.2 • More problems and more misconceptions

I have already analysed many problems, mostly improperly addressed or unacceptably solved. There are even more of them. Let's go through the rest of what I consider important, so we can properly approach the design of other highly secure devices, starting with OTP encryption devices.

4.2.1 • Loose definitions

Not only problems to solve, but even some crucial terms and procedures important for cryptography and electronic security are not defined or recognised well enough. The article

7 My **half-Siamese** lab assistant (and carnivorous predator), also mentioned in chapter 1. Extensive on-going research conducted by many Elektor experts confirms the overall superiority of a felis catus over an average homo sapiens in this role. Less expensive in any aspect, easier on human nerves, inflicts less damage to lab equipment, better hygiene, more effective against flies, rats, snakes and cockroaches, more security-conscious, etc...

8 Not an ICBM (*Inter-continental Ballistic Missile*), but a plain **spear**. The ICBM name is misleading, since it travels in ballistic regime (which means without its own thrust/propulsion) for the major part of its trajectory, but it still needs rocket engines to take off and accelerate fast enough to reach a sub-orbital altitude before diving back down on its target. A spear is 100% truly ballistic weapon, and also the **longest-range** weapon available when invented, and remained so for more than 200,000 years until the advent of **bow and arrow**.

9 КГБ: Комитет Государственной Безопасности, or Committee for State Security, the USSR intelligence agency. The most notorious departments were "**Directorate S**" ("**Управление С**"), operating a network of highly-trained deep-undercover illegal agents in the West and "**Laboratory 12**" ("**Лаборатория 12**"), a chemical lab for research and production of undetectable poisons used for covert assassinations.

[21] may appear unusual, but it makes sense since computer security is difficult to define and measure. It can't be quantified and hence exactly scientifically tested, so a question of whether it can be considered a science at all arises logically!

Locksmithing is a branch of mechanical engineering. It has been around for much longer than any kind of electronic security, for more than 1000 years. Its security standards are well defined. Why? The reason is that practically all the tools and methods of attacking a lock and a safe **are well defined**. Locks can be **picked**¹⁰, bypassed (one of the most widely used methods is **shimming**¹¹), **bumped**¹², drilled, broken, or defeated by many other methods well known to expert locksmiths. Besides this, the doors can be force-opened using a crowbar, sledgehammer, acetylene torch, **thermal lance**¹³, **thermite**¹⁴, **dynamite**, etc. Once again, practically all methods of attacking a mechanical lock are well known, so it is relatively easy to certify a particular lock, door, or safe like in one of the following examples:

1. The lock can withstand a **lock picking** attack by an experienced locksmith for at least 30 minutes.
2. The lock is **bump-proof**. Bumping it is nearly impossible. Achieved by making the pins of significantly different mass (using different alloys), loaded by springs of different stiffness. This way, the dynamics of individual pin motion are different, so they won't all pass the **shear line**¹⁵ simultaneously when bumped.
3. The door can withstand an **acetylene torch** attack for 60 minutes or a **thermal lance** attack for 30 minutes.
4. The safe can't be broken by an explosion of 10 standard sticks of strong 60% nitro-glycerine dynamite.

10 lock picking: opening a lock without a key, using **lock-picks** (for setting individual pins, "picking") and/or rakes (for setting multiple pins at once—"raking"). A certain kind of a **side channel** (like sounds or micro-motions of lock elements) is necessary for it to work. Picking a lock is like timing attack or power-analysis attack on a Smartcard (also based on certain side-channels). The time needed is increased approximately **linearly** with number of lock pins (or bits of Smartcard's PIN), unlike a corresponding brute-force attack (trying all the possible keys).

11 shimming: opening a door by engaging a door latching mechanism, without defeating a lock itself. Performed by inserting thin tools (e.g. a so called "**Slim Jim**") through gaps.

12 bumping: opening a lock without a key, by hitting all the pins with a specially designed bump-key. This makes all the pins jump, thus releasing the lock for a fraction of second, so its cylinder can be turned.

13 thermal lance: an iron-alloy (aluminium and magnesium usually added) rod burning in a stream of pure gaseous oxygen. Acetylene torch is needed for ignition. More effective than an acetylene torch alone, because it uses hot **liquid** (molten metal and metallic oxide mixture) instead of hot gas to transfer the heat to the target.

14 thermite: an incendiary mixture of metallic powder (usually aluminium) and metallic oxide (usually iron oxide). When ignited, releases a stream of hot molten metal (iron), while the combustion is supported by oxygen contained in iron oxide, yielding also aluminium oxide as the secondary product of the reaction. Unlike the thermal lance, this method doesn't need dangerous and difficult to handle gaseous 100% oxygen.

15 shear line: a line between a **plug** (rotating, moving part of a lock), and housing (static part). At least one pin across the shear line is needed to prevent rotation and opening.

The security of a mechanically protected system can be more or less precisely quantified and measured and confirmed by repeatable experiments - hence **locksmithing** can be considered a science since the degree of security it provides can be **measured**.

Let's try to make a similar precise distinction in the world of computer security by comparing a **DoS**¹⁶ attack and a buffer-overflow attack (see 2.3).

The problem with computer security is that certain attacks and resistance to them can't be precisely recognized and defined. If a certain system is well designed to defend against a buffer-overflow attack, how will it do against a DoS attack?

At first glance, these two attacks are different. A buffer-overflow attack is more elegant, sending only one request consisting of one precisely formed sequence of bytes to the target system. Maybe it will just smash the stack and cause a reset. Maybe it will divert the server to some undesired action or even subvert it to the attacker's full control by executing the injected code.

DoS is a brute attack, which floods a server with multiple requests... Wait a second, what if Mallory tries to inject 0x00, 0x00, 0x00,... to the stack (in an attempt to effectively cause the system to reset by returning to address 0), but doesn't know much about the memory structure of Alice's system, so she will simply try injecting megabytes of 0x00? Maybe she won't succeed in overflowing the buffer and force-writing to the stack, but Alice's computer will be flooded by an incoming stream of 0x00 bytes anyway. What if it crashes or even simply denies legitimate requests from Bob busy with processing garbage 0x00 stream coming from Mallory? What was it then? DoS or buffer-overflow?

How to precisely define and quantify a system's resistance to DoS and/or buffer-overflow attack? There are numerous other examples when even a certain attack on a digital system simply can't be precisely defined, which logically means a system's resistance can't be defined and hence measured! How many gallons of acetylene, how many locksmith's man-hours, or how many sticks of dynamite... - this is a way easier to define and measure.

4.2.1.1 • Let's try to define encryption strength...

How to define it? How to measure its strength? What level of encryption do I need for a particular application? The best you can do is express a total number of all possible combinations of bits defining the encryption key. It is relatively easy to express mathematically. If a system can have 40000 different keys, this is cca. 15-bit, because $40000 = 2^{15.29}$.

The problem with this as a measure is that it only tells you how resistant it is to a brute-force attack. It is easy to calculate how long it takes to test one key. Multiply it with a total

16 DoS - **D**enial **o**f **S**ervice. Executed by flooding the target system by too many meaningless requests. An example is dialling a certain phone number repeatedly and hanging up, or sending a request to a web server every 10 milliseconds. It won't always cause the server or phone exchange to crash because of overload, but may at least succeed in denying the service to legitimate users.

number of all possible different keys, and you will know how many computers would need to work for how many years to test all combinations. The result you get this way is practically useless. One-time pad encryption can never be cracked by pure brute force. Does it mean that it is unbreakable (remember VENONA)? How many codes have been defeated like this in the past? Defeating Enigma was possible by quickly reducing key space, assuming that a U-boat commander sent a few easy to guess words, like “CONVOY DESTROYED” after a successful attack. You could never defeat a mono-alphabetic cipher in the 14th century by brute force, so Ibn al-Durayhim found a faster method. How to express strength of a mono-alphabetic cipher? It depends on the particular language used more than on the length of its alphabet. If a hypothetical “cryptographically perfect” language had an equal frequency of occurrence of all letters, it would be impossible to quickly crack the code.

One very important question when choosing an appropriate crypto-method, routinely overlooked, is “**For how long does it need to hold?**” This was already analyzed in 4.1.3. Sometimes 10 minutes is enough. Sometimes one would prefer several hundred years. This can be compared to the mechanical gradation of locks and safes (e.g. “capable to withstand a thermal lance for 30 minutes”). We will see later when analysing the “PGP Affair”, how easy this was to misunderstand in practice.

4.2.1.2 • What is encryption, and what is not?

Now it is about time to define encryption, so we can tell the difference between encryption and some other procedures (scrambling, obfuscation, steganography, etc...) that only resemble it.

The Oxford Dictionary states “*to encrypt - to put information into a special code, especially in order to prevent people from looking at it without authority.*” This isn’t very precise, and we will soon see why. I checked a few other sources (like e.g. Merriam-Webster Dictionary). They weren’t any more precise either. Now we will see why more precision is required.

First of all, it’s important to distinguish between **encryption** and **scrambling**. We will then address methods like **obfuscation** and **steganography**. Many procedures which can’t be considered as encryption may resemble it at first. This can lead to some products being falsely advertised as encryption-enabled, although they are not.

Scrambling a signal means simply re-arranging it in **time-domain** or **frequency-domain** - like cutting it into small fragments and changing their order. A few examples are:

1. Simple voice-scrambling works in a way to digitally record e.g. 500ms of speech to a buffer, and divide it into 25 segments of 20ms which are then shuffled (re-arranged in time). The total number of combinations is then $25! = 1.55e+25 = 2^{83.68}$. This, of course, doesn’t mean that this system is as strong as 84-bit encryption. Not even close! The receiver needs to re-arrange the segments in the correct order to make them legible.
2. Re-arranging the order of scan lines on a CRT monitor as a method of protection against van Eck phreaking is also a type of scrambling (anti-TEMPEST).
3. Printing every single line of text with a dot-matrix printer with several passes of printer

head (instead of just one) and random order of printing individual dots is another type of anti-TEMPEST scrambling.

4. Spectrum-inversion scrambling (this is done in the frequency domain, not time-domain) was a popular analogue method of concealing telephone conversations in the old days. It was also used in the late 1980s and early 1990s for concealing TV broadcast transmissions. Only paying subscribers, provided with proper analogue hardware could de-scramble the signals.
5. Paper-shredders are devices that perform the **mechanical** scrambling of data.

Scrambling methods (e.g. simple time-domain voice-scrambling) may also involve setting a proper key on both sides (like for encryption). The main difference between scrambling and encryption is that scrambled information is **partially legible** (in fragments), while well-encrypted information is not. A segment of 20ms of human speech is still plaintext speech, although not easy to recognize using naked ears. A single scan line of a scrambled CRT monitor is still the same as the original. Spectrum-inversion scrambled human voice still resembles human speech, and individual words are still easy to separate. Partially successful de-scrambling will yield partially legible plaintext, which is not true for good encryption. Scrambling may be a lot simpler than encryption, and is still very useful, especially if combined with other methods to enhance security, but it can't be considered to provide as much protection as encryption, especially if used alone. A typical dirty marketing trick is to advertise scrambling as encryption. Alice and Bob must be aware enough not to fall for it.

Good examples of defeating scrambling protections are the **Elektor Filmnet TV decoder** (actually descrambler) from 1989. (re-generates analogue TV PAL sync signals hidden modulated in an extra sub-carrier) and re-arranging the paper-shredded documents in the USA embassy in Tehran, after it was occupied in 1979. The shredders used were a very cheap type. Secure destruction of data wasn't taken very seriously.

As explained in 1.2.2, **steganography** is hiding secret messages inside other, much longer messages. This is easy to distinguish from encryption and scrambling. **Obfuscation** may be similar to steganography but refers to hiding a program code or even parts of hardware. Steganography, on the other hand, is about hiding messages. It uses a **legible, meaningful long message** to hide a shorter (plaintext or encrypted) useful message, unlike when obfuscating a program code, a short useful code is **obfuscated by long useless code**, which can be completely meaningless garbage. Another way to obfuscate machine code (originally used actually to save scarce ROM memory space in 1970s-1980s) is to use parts of **look-up tables** (parts of ROM not meant to be decoded by CPU as **program** - machine code instructions, **apparently** used only to store arrays of **data** constants and e.g. fixed plain-legible ASCII text messages to be displayed on a VDU or a printer) also as **hidden** segments of a program to be executed when needed. For example, if the CPU is a Zilog Z80, and the text to be displayed on VDU, stored in ROM is "HELLO. " If a program jumps to the address of the character "H" (i.e. loads "H"-0x48 to instruction register IR, not to some general-purpose register like A,B,C,D...), the following hidden sequence of assembly instructions will be executed:

LD C,B ; LD B, L ; LD C,H ; LD C,H ; LD C,A ; LD L,0x20

Code obfuscation can be done on machine code, but also HLL source code, especially if the HLL used is **C** (the only serious secure crypto programming tool of trade, besides assembler). There is even an international contest in writing **obfuscated code in C**, mentioned in [20], held every year. The highly flexible structure of C source code enables obfuscation on a source-code level. The following is an example of a “grammatically correct” C, with no syntax errors!

```
int i;main(){for(;i[“<i;++i}{--i;}”];read(“-’-’-’,i+++”hell\
o,world!\n”,’/’/’/’));}read(j,i,p){write(j/p+p,i---j,i/i);}
```

Besides the program code, obfuscation can be also done on electronic circuit schematics (i.e. on a hardware level), as you will see later in chapter 5.

4.2.2 • Symmetric and asymmetric encryption

The basics of symmetric and asymmetric encryption were already mentioned several times in this book. We will now summarise because it is necessary to understand the events related to the next real-life crypto-affair to be analysed. As we will see, many events were simply results of bad interpretations following loose definitions.

Methods like AES, one-time pad, mono-alphabetic cipher, ENIGMA code, and many other are **symmetric**, meaning that Alice and Bob use the same key for encryption and decryption. They need to meet securely to exchange the keys or find some other method (will also be addressed later in this chapter). RSA and El-Gamal are **asymmetric**. Alice uses one key (Bob’s public key) for encryption, and Bob uses the other (Bob’s private key) for decryption. This works because of certain advanced mathematical procedures discovered in the 1970s. Bob can publish his public key to everybody (like on his web page), and receive their encrypted messages that only he will be able to decrypt and read.

The RSA (Rivest-Shamir-Adleman) asymmetric encryption is based on fast multiplication of two large prime numbers $n=pq$, and very slow or near impossible factorisation of the semi-prime product n . Number m is a part of the plaintext message (or only one letter), number c is the corresponding ciphertext. Number e is the encryption exponent, d is the decryption exponent, n is the modulus. After p , q and e are chosen, d is calculated. Here is the procedure:

1. Choose two prime numbers p and q .
2. Calculate $n=pq$.
3. Calculate $t(n)=lcm(p-1,q-1)$. Function $lcm(x,y)$ is the “least common multiple” of two numbers, the lowest number which is a multiple of both. It will equal their product if they are mutually prime (they share no common prime factors). Examples - $lcm(24,28)=168$, but $lcm(35,72)=35 \times 72=2520$.
4. Choose e such that it is mutually prime with $t(n)$.
5. Find d such that $(de) \bmod(t(n))=1$. This is the most complicated calculation required to set up a pair of keys (public and private).

What is important to remember is when Bob completes these 5 steps, he will **publish n and e** to Alice (and also everybody else) as the **public key**. He will keep p, q, d , and hence $t(n)$ secret, to himself, as the **private key**. If p and q are large enough, it is very difficult to factor n (publicly known) to find p and q . Both p and q must be known to calculate $t(n)$ in step 3.) and d in step 5.), which then fully reveals the private key. **RSA encryption is as strong as n is difficult to factor - this is the most important fact to remember!** Encryption and decryption are done using the following equations:

$$\text{Encryption: } c = m^e \bmod n$$

$$\text{Decryption: } m = c^d \bmod n$$

Since n and d are large numbers, the equations are computationally demanding. The usual encryption procedure is that Alice encrypts only a one-time **session key** with Bob's public key and sends it to him. Bob then decrypts the session key with his private key, and Alice and Bob then communicate using some symmetric encryption (e.g. AES), only for that session. The AES key will thus be exchanged remotely (even if Alice and Bob have never met and even if they don't know each other) and remain a secret.

Besides encryption, RSA can also be used for **digital signatures**. Maybe Alice doesn't need to encrypt the information but needs to authenticate the message she sent to Bob, so he can be sure she sent it. This was done using handwritten signatures and seals (like signet rings) for centuries. The digital signing process works like this:

1. Alice writes a message in plaintext. She can also encrypt it if needed.
2. She calculates a hash of the message, using e.g. SHA-1
3. She encrypts the hash with RSA, using her private key. The encrypted hash is her digital signature of this particular message.
4. She sends the message along with the digital signature to Bob.
5. Bob decrypts the signature using Alice's public key.
6. Then he calculates the SHA-1 hash of the message and compares it with the decrypted signature. If they match, everything is OK, the message is well authenticated to Alice.

RSA, the same as any other method, is not without pitfalls, even if the mathematics behind it is perfect. Now we will see that many things can be misunderstood and go wrong. Alice and Bob can't afford too much trial-and-error, given that there is not much room for it in their line of business...

The first problem is choosing large prime numbers. Checking if the number is prime by "brute force" (i.e. dividing by all known smaller prime numbers) is way too slow. More advanced mathematical algorithms for generating and testing potential prime numbers have been invented, of course. There are also pre-computed lists of prime numbers. About 10^{17} prime numbers have been listed from 2 till cca. 4×10^{18} . Here comes the first problem: an 18 digit prime number has only 60 bits. The product of two such prime numbers ($n=pq$) is then 120 bits long. This list is in order of 1 million TB (terabytes)! which is not very practical to store on a HDD. A 120-bit RSA modulus wouldn't be much use anyway. In [22] you will see why. Nice monetary prizes have been offered, for factoring some big semi-prime numbers. A 330-bit semi-prime can be factored on one average PC, using a

good algorithm **in a few hours**. A 426-bit semi-prime was worth only 100\$ prize. A 576-bit semi-prime was awarded 10.000\$. The largest successfully factored semi-prime was 829 bits long. A prize of 100.000\$ was offered for a 1024-bit number. There has been no success as of yet (February 2021).

This refers to teams of top-class mathematicians working on very hi-tech hardware, using very advanced algorithms. It takes 1 million 1TB HDDs to store the prime numbers needed to generate a uselessly weak 120-bit RSA key (i.e. n number, or modulus). The minimum required for any decent RSA encryption is 512-bit, based on the information in [22].

A procedure to find 256-bit long prime numbers p and q is far from trivial. The same goes for calculating d , the decryption exponent in step 5. Many things can go wrong even during these initial steps! How much does Alice know about this procedure on an average RSA program running on her PC? She can read the p and q generated, but how can she reliably test if they are prime numbers? RSA programs usually use some procedures controlled by an RNG (usually a PRNG) to pick these numbers. Do you know anybody who has tested them reliably?

There are more problems. Remember when I said that m and c were **parts** of plaintext and ciphertext messages, encoded in appropriate numbers? I've seen dangerously weak RSA implementations where " m " were **1-byte numbers** representing a single ASCII character of a plaintext. This was done probably because RSA arithmetic with large numbers was too computationally demanding. If a single plaintext letter m always encrypts to a single corresponding ciphertext number c , what has this RSA become? Any better than a **mono-alphabetic cipher**, possible to crack with 14th-century math? Numbers m should be at least 5-6 bytes long to ensure reliable encryption if encrypting a plaintext directly with RSA. This is yet another reason why it is better to use RSA to encrypt only a one-time session key (a meaningless sequence of bytes), to be used to encrypt the message with some symmetric encryption like e.g. AES.

Public key encryption methods like RSA are extremely important and valuable. The modern networked world would be impossible without them, but like any other crypto-tool of this world (including OTP, of course), they can also fail easily if improperly implemented and handled.

4.2.3 • PGP affair

As you can see, generating RSA key pairs can be done "manually", but requires a lot of mathematical computation. The prime numbers p and q must be chosen first. They must be prime and very large (at least 256-bit) to generate a strong key. Number e is not so critical. Finding d then requires a lot of calculations. Calculating integer division remainders with big numbers (when encrypting and decrypting) can also be much accelerated using clever maths.

With all this in mind, Phil Zimmermann made the PGP program in 1991. Other improved versions by many other authors have later followed. He made the procedures to calculate

key pairs based on random events (like mouse motion performed by the user). He then used a so-called base64 encoding to put the keys in a textual form more appropriate for copying and posting. The public key file consists of n and e numbers, encoded in **base64**¹⁷, along with some extra formatting bytes. The private key file is usually longer, containing n, p, q, e, d , and some extra auxiliary variables. When a 512-bit (number of bits of modulus n) key pair is generated, the two textual files defining the key pair will look something like the following:

```
-----BEGIN PUBLIC KEY-----
```

```
MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAMh94q8NSNsioLFqHFOZaZVcrT4g4KiDKSwRy-  
DgAHRBnwpTXpyfstEsMNTx2wCrE3tKJuZdrsSwnQD0LQzGy2UCAwEAAQ==
```

```
-----END PUBLIC KEY-----
```

```
-----BEGIN RSA PRIVATE KEY-----
```

```
MIIBPAIBAAJBAMh94q8NSNsioLFqHFOZaZVcrT4g4KiDKSwRyDgAHRBnwpTXpyfst-  
dEsMNTx2wCrE3tKJuZdrsSwnQD0LQzGy2UCAwEAAQJBAljnJPYhnSE9ut2rFmKjf-  
Cxol0e4TY/j2CZmkhBjS3VaH1BH68HvDW88GAxA1oQwPounLF3r8K6e3WWPL+SE/  
mECIQDoBngXJI73Igj9684+V7d0LG+3oPskyu0+5+U7CFGtTQIHAN01SqPJ1rEy/Lzwjy6usN-  
LOCgYTT648JZYwi+QREmp5AiBxR/9H5Ux7sj1Lo95NJ8xk+bfVop4bZ4wPY27StIwUHQIHANK-  
CJ3bcY8onqmqnRqr3JWDHILTDZcN6i5SMUeQM9wiJAiEakfn7ZKZeJmwf+I2LJ4DxDPaRd7FeC-  
QLuaN15rSCsGpg=
```

```
-----END RSA PRIVATE KEY-----
```

Similar formats are used for storing digital signatures in plain text files (*.txt) as well. The private key file must be stored securely on Alice's computer, so she also defines a password to encrypt and conceal it. The private key is revealed only for a short time when decrypting a message from Bob. Phil Zimmermann also automated the process of generating a session key for one-time symmetric encryption of a single message. This way he provided people with a user-friendly crypto tool.

Besides the aforementioned pitfalls, even more became apparent after the PGP affair. What happened was that the encryption PGP used was too strong to be brute-forced in a reasonable time. At the time, any crypto-tool with a 40-bit or longer key was legally treated as military equipment (like weapons or ammunition). Phil Zimmermann was accused of something like exportation of arms and endangering state security. The charges were later dropped, but the media blew it out of proportion as always (such as **"TERRORISTS will be**

¹⁷ base64 is a data format, for representing raw binary data in a more compact and legible way than for example, hexadecimal numbers. The data is first divided in groups of 6 bits (64 possible symbols). Each group of 6 bits is replaced by 1 symbol defined by base64 standard look-up table - 26 lowercase English letters, 26 uppercase, 10 decimal digits, and symbols "+" and "/". Symbol "=" is used for padding and has no value. Based on base64, Satoshi Nakamoto defined base58 for representing the Bitcoin addresses, similar to base64. Some "ambiguous" characters are easy to mistake (e.g. "o", "O", and "0" may look the very similar in some fonts, the same goes for "1", "l" and "I" as well). Chars "0", "O", "I", and "l" are hence excluded, along with "+", "/", and "=".

able to secretly communicate, and we will be helpless!!”), and led many people to believe that PGP encryption is unbreakable, without even considering the length of the key (number of bits of n modulus), let alone all other previously mentioned factors.

I knew people familiar with computer technology (programmers or electronic engineers), in the mid-1990s, who stored files with confidential information on SRCE (Computer Centre of Zagreb University) public servers encrypted with only 56-bit PGP. They considered it unbreakable, which led me to believe that the over-inflated frenzy was created as a part of a conspiracy. Even educated people became **less security-aware** (PGP was designed to give a simple crypto-tool to the people, and hence make them **more security-aware**), and more prone to leak critical information themselves! Now we all know it was possible to brute-force a 56-bit key in the mid-1990s in a reasonable time. Furthermore, Eve could simply have copied the files and tried to brute-force them 5 or 10 years later, with much faster computers and algorithms, and still extract useful secret info.

4.2.4 • Quantum computers

They are the technology of the future. The way they define and process information is fundamentally different from classic digital computers, in the same way quantum mechanics differ from classic Newtonian mechanics. Let's go through the basics:

1. A classic digital computer represents information in **bits**. Every bit can have a defined distinct value 0 or 1 at a given pulse of the computer's system clock. A quantum computer represents information in **qbits**. Every qbit can exist in a certain superposition (or ratio) of 0 and 1 states, or to express it differently, can have a **certain probability** (any real number between 0.0 and 1.0) to be in state 0 or state 1. In a classic computer, an n -bit word can have 2^n possible states. In a quantum computer, an n -qubit word can consequently have a practically **infinite** number of possible states.
2. A qbit is related to a quantum physical property of a single particle possible to be expressed in two states, like polarisation of a photon (horizontal or vertical) or spin of an electron (up or down).
3. Quantum computers follow all the principles of quantum mechanics. One of them is **Heisenberg's uncertainty principle** which states that an exact quantum state can't be exactly calculated even if all the conditions are known. Only a certain probability can be estimated. The state can be measured, which can't be done without disturbing that state! This means that the state of a qbit can be measured only once (unlike the state of a classic digital bit which can be measured many times, without changing its state). If qbits related to for example, a photon polarisation are used by Alice and Bob for their secret communication, they can't **be monitored by Eve, without being changed**. Therefore Alice and Bob will quickly detect eavesdropping (impossible with classic digital computer systems).

The principles of quantum mechanics are highly counter-intuitive. Because of a potentially high amount of information contained in a single qbit, quantum computers are much faster

in processing information than classic computers. On the other hand, at the end of the 20th century, it was still difficult to make a quantum computer add two numbers reliably. Nowadays they can multiply and even factor some 2-digit numbers ($21=3\times7$). Advanced procedures like **Shor's algorithm** (for fast factorisation of large numbers) were invented in the 1990s. Quantum computers reliable enough to perform these procedures were not yet available. Shor's algorithm is much faster than for example **"General Number Field Sieve"** (GNFS - an algorithm also used for factorisation, much faster than brute-force factorisation), but GNFS can run on a classic computer, and Shor's algorithm would require a good quantum computer. Needless to say, as they evolve, they could (in theory) eventually render encryptions like RSA useless, if a 300-digit semi-prime number could quickly be factored.

They could also make a much more difficult procedure of **reversing a hash function** possible, thus killing for example Bitcoin or similar crypto-currency systems, including every other security which relies on a hash function.

Quantum computers are also usually misinterpreted and widely misunderstood. Yes, they have the potential to greatly speed up complicated and demanding math calculations (like factorisation), but these procedures can also be performed on classic digital computers, only much slower.

On the other hand, they can't make miracles happen. When I was 9 years old, I approached a Sinclair ZX Spectrum with great respect because at first, I considered it a "miracle" box, capable of everything seen in SF movies of that time (e.g. zapping its "master" if mistreated). Some misunderstandings of quantum computers go so far that they are considered capable of regenerating certain random effects, like for example, electronic noise. If they could ever become capable of doing this, it would be possible to regenerate every noise and thus easily cancel noises out. This would lead to infinite-range communications because it would be possible to amplify every extremely weak signal after cancelling out every noise. This way, even OTP encryption would become useless, because quantum computers could re-generate any possible random number sequence, created anywhere in the universe. Predicting and re-generating true random numbers is something quite different than performing well-defined mathematical calculations (like addition, multiplication, and factorisation) faster. These fundamentally different tasks should not be put in the same basket.

4.2.5 • Reversing an implication and T-com payphones

As we have seen up until now, many low-tech systems can be designed and used effectively to increase security for Alice and Bob. Payphones are a relic of the past. Some are still operational, but as far as Alice and Bob are concerned, they are not a secure low-tech system (as explained in 1.1.4) so they better forget about using them. Why am I mentioning payphones again? Well, there is a very good reason...

Implication is a logical operation. $A \Rightarrow B$ means that statement A implies statement B or that statement B is a logical consequence of statement A. This is not only a human-readable mathematical equation but a process that **intelligent** beings routinely perform

when **making decisions**. In the case of a cat and a mouse, it works something like *"If I am hungry and cheese is detected and a cat is not detected nearby then I am OK to approach and eat that cheese."* Any mistake in this logical equation is a matter of life or death for this mouse. It will get clearer when we apply a similar conclusion for some crypto method used by Alice and Bob. How do they conclude that a certain method is secure or not? We will get back to this shortly...

If $A \Rightarrow B$, then it is logical to conclude that $\neg B \Rightarrow \neg A$. This means that if B is not true, A is consequently not true. Fine. We can illustrate it with a few examples.

A="It rains.", **B**="Streets are wet.", **$\neg A$** ="There is no rain.", **$\neg B$** ="Streets are dry."

In this case, $A \Rightarrow B$ means *"If it rains, streets are wet."*, and this is a true statement. The statement $\neg B \Rightarrow \neg A$ (which is a direct corollary of $A \Rightarrow B$) then means *"If streets are dry, there is no rain."*, which is also a true conclusion. Everything has been clear until now. The problem is that people often make the following mistake: based on $A \Rightarrow B$ they correctly conclude $\neg B \Rightarrow \neg A$, usually followed by **$B \Rightarrow A$ which is false!** In this case, it would be *"If streets are wet, it rains."* This is called **"reversing an implication"**. Well, this doesn't have to be true. Maybe a pipe has burst, and water flowing out has made the street wet. Maybe it snowed yesterday and now the sun is shining and melting the snow, so the streets are wet. Again no rain was involved. Maybe the streets are being cleaned, flushed with water from hydrants. There are so many ways to make streets wet, without rain. This is a simple example, but...

We all know that every cryptosystem needs to be evaluated by several independent experts. If **any** of them finds a feasible method to crack it, he will normally describe and publish it. This way everybody else will be able to test it. If it works as described, the attack method is proven and the cryptosystem will not be considered secure. This is relatively simple to understand. On the other hand, if nobody can find a method to crack the protections in 5-10 years, we can consider the system reasonably secure because a feasible method to crack it hasn't been published.

"If a crack is published, the system is not secure." - CORRECT, $A \Rightarrow B$

"If the system is secure, a crack is not published yet." - CORRECT, $\neg B \Rightarrow \neg A$

"If a crack is not published, the system is secure." $\neg \neg$ FALSE!!, $\neg A \Rightarrow \neg B$

The problem with this line of reasoning is that normally the method to crack the system is published when discovered, but it also doesn't have to be. There have been examples of this exception to the rule, and I know of at least one for sure. After the downfall of Yugoslavia in 1990, its national telephone company (called "PTT") got split into its former federal republics. Croatian Telecom was quickly acquired by Deutsche Telekom AG. They phased out coin-operated payphones and introduced chipcards. Most of the time (until today), Croatian payphones have used SLE4436 chipcards, also known as "Eurochip 1". Slovenian payphones used a very similar SLE5536 or "Eurochip 2". Payphones around the world nowadays accept credit cards. Some still work with stored-value cards, and some can accept both.

Let's clarify some terms first. You can start by reading article [23] which explains the basics of Smartcards. Most smartcards today are microprocessor-microcontroller (MCU) cards, although there are still some memory-only cards. Most MCU cards (like for example the "**Funcard**" - containing an Atmel AT90S8515 MCU and some extra AT24C EEPROM) can be programmed with an appropriate assembler, or the assembly code can be generated with a C-compiler for example. Others have more complex microcircuits, like the ZeitControl "**BasicCard**". This contains an operating system along with a versatile BASIC interpreter, so it doesn't need compiling. The card is simply loaded with a BASIC program in source code. A "BasicCard" can easily be programmed to work as an ATM credit card. Although BASIC interpreters are generally slow, the speed is usually not critical for this application.

Credit or debit bank cards don't store the amount of money themselves. When a transaction is requested, they contact a bank server to check, because the bank server stores the balance. Credit cards are hence **not stored-value cards**. "Eurochip" phone cards **are stored-value cards**. When they were introduced (the late 1990s- early 2000s), payphones could not quickly contact a server online, so the system relied on **disposable** stored-value cards. The amount of credit left is stored on a card itself. Although there is a possibility of reverse-engineering a phonecard and making free phone calls, the phonecard payphones require less maintenance than coin-operated payphones (less mechanical problems and no need for collecting coins).

Now we are coming to the main problem with these phone cards. The cheapest phone cards (still available today on kiosks in Croatia!) store only 2 EUR worth of credit. After 2 EUR worth of phone calls are made, the card is **thrown into a trashcan**. How much are the microelectronic circuit and the card's plastic body worth? Back in the year 2000, an average 8-bit MCU was much more expensive. What is the point of selling a phone card with 2 EUR of stored credit implemented on a **disposable** MCU worth at least 4 EUR? The microcircuits of a phonecard had to be extremely cheap and simple (no CPU-MCU at all), but still capable of some basic digital authentication, to make them more difficult to make counterfeit.

"Eurochip" phonecard authentication works in a way that a payphone generates a 48-bit pseudo-random challenge to the phonecard, to which the phonecard responds with a 16-bit response. The response is then checked by the payphone. If the response is OK, the phone card is considered valid, and the call may proceed. "Eurochip" uses a CRC function to calculate the response. CRC can be implemented as a serial shift register with several XOR gates, without any CPU. Anything more complex would have been too expensive to be sold in 2 EUR disposable tokens.

CRC¹⁸ functions are linear and very simple to implement. They are good at detecting accidental errors, but they are not good for authentication. Being mathematically linear, they are easy to reverse. If Mallory wants to plant a false message, she can quickly calculate a few bytes that need to be changed, to make the CRC checksum bytes match. This is explained in [6]. This is why non-linear hash functions (MD-5, SHA-1, SHA-256) were invented because they are difficult to reverse. It is difficult to change a message at will, while still making the hash checksum match. This is why they are good for authentication.

Needless to say, the Eurochip cards were quickly cracked, reverse-engineered, and counterfeited after they had been introduced. Just out of curiosity, I bought a 2 EUR phone card in 2015, and it was still the same old SLE4436! Since then many more payphones have been removed from the streets, but to my surprise, many new types have been set up, with large LCD touch-screens and WiFi hotspots. They are still accepting the same phone cards today!

So, what's the catch? If you try searching the web, you will easily find many "warez" sites with cracks for all sorts of commercial software that would have to be bought legally. You will find procedures to crack WEP or WPA WiFi passwords. You will easily find a torrent to illegally download practically any pirated movie (e.g. on a site like Pirate Bay). Articles on performing all sorts of obscure TEMPEST attacks can be found. You will easily find a lot more critical information, but I couldn't find any webpage or an article with a procedure on how to reverse-engineer a "Eurochip" card. The only possible explanation is that these pages are being systematically removed from the web, similar to the torrents of new blockbuster movies while they are still making money in cinemas.

"Eurochip" is still being used up to this day, and cracking it and making counterfeit phonecards is not lucrative anymore in the world of cheap smartphones. Maybe an article about reverse-engineering it could be found somewhere on Dark Web, I haven't tried that. We can (for all practical purposes) consider that such an article **hasn't been published** up to this day. Does this mean that a "secure" payment system based on a linear CRC function for authentication, performed locally, with no network access, is secure enough to be trusted? This is one example that I know about, but there are surely many others. Can you think of some other similar "exceptional" events? Maybe my line of reasoning is wrong? Or maybe these payphones are intentionally being used as "honeypots"¹⁹, for detecting unwary spies, hackers, and counterfeiters?

18 CRC (Circular Redundancy Check): a simple method for calculating redundant control bits, usually used for checking communication errors. The first and simplest method ever used was **parity check** (1 single check bit generated by XOR-ing a stream of bits in a message). Then the **LRC** (Linear Redundancy Check) was invented, a method which generates 1 control byte, by XOR-ing bits in message bytes (all the bits of the message at position 0 XOR-ed, result stored at bit 0 of control byte, and the same for all the 8 bits). CRC was invented as a more advanced method, works by shifting a message bitwise through a shift-register (e.g. 32 bit register for CRC-32 function) and XOR-ing some of the shift-register bits with incoming input bit.

19 honeypot: a bait intentionally set to attract various types of villain.

The moral of this story is that testing a certain system by several independent experts, and reading their reports is important because this is the only way to effectively evaluate security. **Even this principle** has exceptions (as we have just seen with “Eurochips”), so Alice and Bob must be extra careful. FER students who annoyed Croatian T-com by making counterfeit phonecards got expelled from the university (a little after we got rid of our last dictator), but I know similar examples from Germany as well. These cases didn’t get much media coverage. Could T-com have found a more embarrassing solution for the company?

4.3 • Black-box cryptography

It is simply bad (for Alice and Bob), widespread, and used everywhere. We have established that already. “Open-box” alternatives aren’t easy to get. Just to remind you again, this book is about helping Alice and Bob, not their adversaries. The following real-life affair will show the extremes that black-box cryptography can reach.

It will also show the west is not the absolute centre of the world’s hi-tech anymore. It wasn’t so even during the Cold War. The East Bloc was still keeping up, they were just not well advertised, and their communist system wasn’t particularly attractive. I learned this very well from my faculty professors at FER who did their specialist Ph.D. studies in the Soviet Union. Regardless of all the well-known pitfalls of communism, their educational system was still very good (confirmed by several Polish, Hungarian, Vietnamese, and ex-Yugoslav former foreign students in the USSR that I have talked to, after the end of the Cold War). Countries like Iran, India, Brazil, China, Pakistan, and many Arab countries have made very significant technological advances in the past 30 years. Western engineers will have to abandon their usual self-absorbed attitudes if they want to remain competitive. Although Croatia is not a part of West Europe in any sense, many of my countrymen also tend to share similar false attitudes, and needless to say, this is not good. Once upon a time, Yugoslavia was the most hi-tech country of the NAM Bloc, and one of 12 countries in the world capable of designing and building a **submarine** (Croatia was even **one of only seven** in 2001, with the automation department of FER taking care of submarine control systems), these attitudes have had some solid foundations, but not anymore. Alice and Bob will now have to learn to look further afield than Europe or North America to make reliable comparisons.

The next story illustrates this very well. The key turning point happened in **Iran**, almost 30 years ago...

4.3.1 • “Crypto AG” affairs

“Crypto AG” was a very well-known Swiss company, dealing with the design and production of all sorts of crypto hardware and software. Switzerland had a reputation as being a neutral²⁰ country. Long-time on-going scandals and affairs linked to this company can be found all over the internet. Only a little while ago, these were on a level of “conspiracy

²⁰ Political neutrality sometimes seems difficult to define. Another country well known for boasting about its “neutrality” was Yugoslavia, of course.

theories” (quotes intended). There is a great difference between “conspiracy theories” and conspiracy theories. Check the back cover. Some of these have existed since the beginnings of human civilisation, in theory, and practice, controlling practically everything that matters in this world. Others are immediately flagged as mental by-products of lower life forms, without any need for logical analysis.

Crypto AG was officially liquidated in 2019, so all of its affairs are now OK to be freely discussed on the internet. The same will happen to “Eurochip”, once it is finally officially terminated.

Crypto AG was selling its crypto devices to more than 120 countries. Click [24] and you will see that they have always had various collusions²¹ with the West Bloc, over many years, since 1955. The ownership of the company was kept a secret for quite some time, eventually revealing BND²² and CIA as its owners. Needless to say, they were selling rigged devices, with weaker ciphers and backdoors to many customers, both civilian and military. This includes Yugoslavia, and Croatia, of course. Buying and using Crypto AG devices was often a condition for many governments to be accepted to various political, business, and military international associations. Only the Soviet Union and Warsaw Pact countries were known for using different cryptosystems.

The first country to publicly expose the conspiracy was Iran. In 1991. Iranian engineers analysed Crypto AG devices used by the Iranian government and confirmed they had been rigged. In 1992, Iran arrested Crypto AG’s top representative Hans Bühler and made it public. Perhaps analysis performed by other experts before would have found the same, but Iran was the first country to step forward and expose the collusion. Crypto AG paid 1.000.000\$ to release Bühler from the Iranian prison. After his release, Bühler publicly confirmed that Crypto AG was selling rigged devices and consequently got fired from the company.

At that time (the mid-1990s) such claims were routinely rejected as “conspiracy theories”.

21 Oxford Dictionary defines “**collusion**” as a secret agreement with an intention of doing something dishonest.

22 **BundesNachrichtenDienst**: West German intelligence agency

4.4 • Elektor OTP Crypto Shield

Up until now, we have seen how easy it is for a cryptosystem to fail. It can fail in an almost infinite number of ways. Even if the mathematics, technology, implementation, and security procedures are perfect, humans operating the system will always find a new way to screw up. Some systems are even intentionally built to fail!

I built this device in cooperation with Elektor after considering all the problems discussed in this book up until now. The Crypto Shield was built as an add-on to the **Elektor UNO R4** (improved Arduino-like system, using ATmega328PB MCU, “B” model with two hardware SPI ports and two UART ports). The UNO R4 may be difficult to obtain, but the same OTP Crypto Shield will operate with a ATmega328P as well, with a slightly different firmware (the other UART port is “**bitbanged**²³”). It uses Vernam’s OTP as the only potentially unbreakable cipher. Figure 4.1 shows the schematics of the shield. If the Elektor UNO R4 is unavailable (figure 4.2), the ATmega328P replacement can be made according to the schematics in figure 4.3. Follow the project link [25] to download my Elektor article and check all the detailed descriptions, including the Labs project page and demo videos.

²³ bitbanging: to emulate certain communication protocol (e.g. UART, SPI, I²C... when not implemented by dedicated hardware) by directly flipping in/out bits using CPU/MCU software.

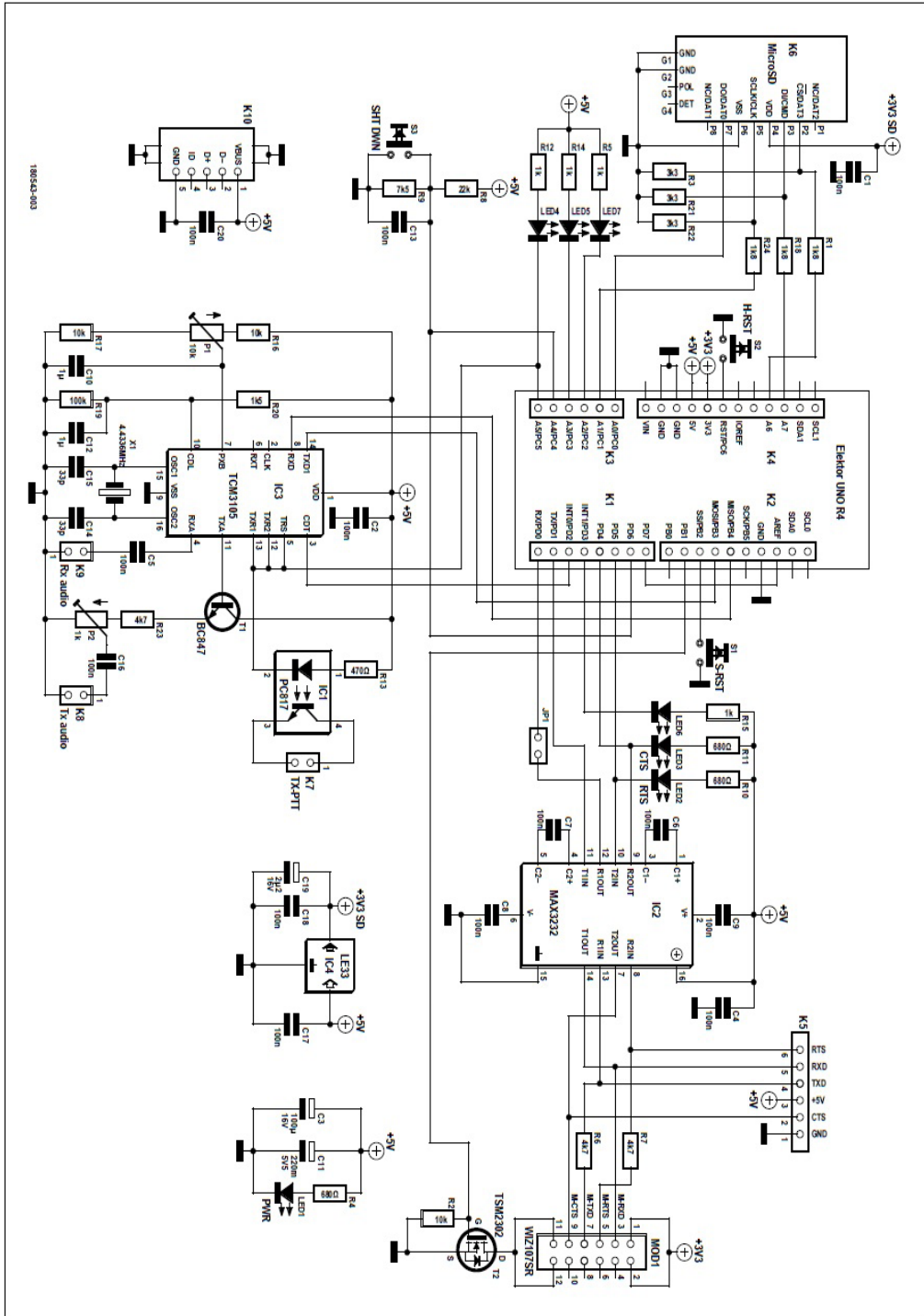


Figure 4.1 - Elektor OTP Crypto Shield schematics

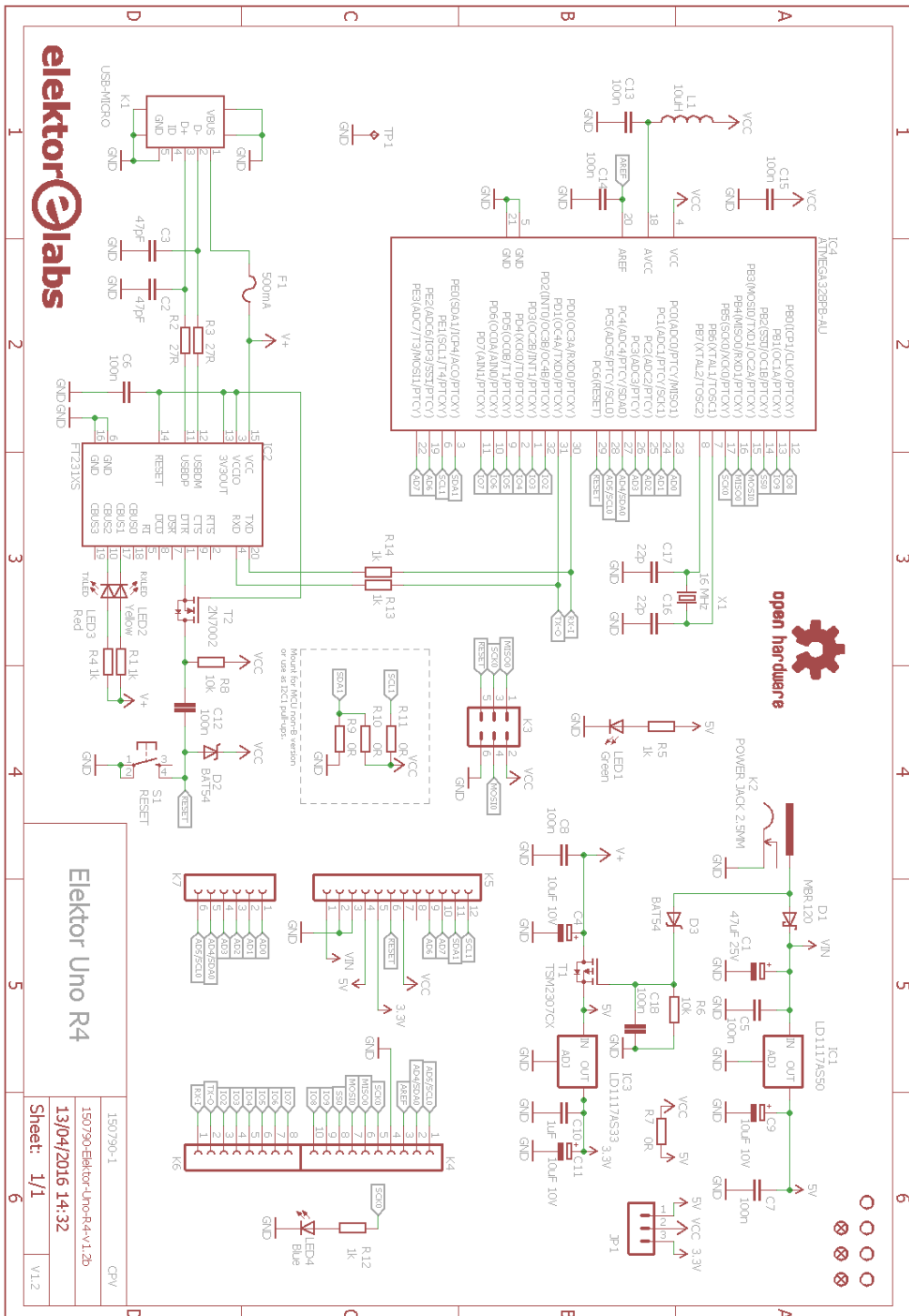
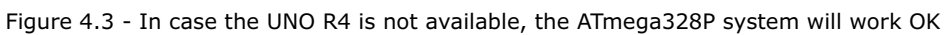


Figure 4.2 - Elektor UNO R4 schematics



The schematics in figure 4.1 can be divided into three main parts:

- The UART-RS232-Ethernet adapters part around IC2 and MOD1 (for interfacing with various terminals, for typing and displaying plaintext messages, and controlling the system),
- The IC3 FSK²⁴ audio modem part (for online **teletype**²⁵ mode using any low quality speech-bandwidth audio communication channel)
- Interface to the SD card used to store OTP keys and files for offline encryption/decryption.

After a plaintext file on the SD card is securely encrypted (in offline mode of operation), it can be copied from the SD card and emailed in encrypted form using any standard insecure channel. Online teletype mode requires a channel capable of transferring 700Hz-2700Hz audio range (low-tech again!). This may be a plain copper twisted-pair cable, an analogue radio transceiver, or any digital audio channel using lossless compression like the G.711. Lossy audio codecs (like codecs used by GSM or Skype, designed to compress human voice) can transfer the human voice, but not FSK modulated signal. Although the bandwidth is the same, the FSK signal frequency shifts much faster than the pitch of a normal human voice. The very narrow band audio communications channel used will additionally mitigate buffer-overflow or DoS attacks by limiting speed, by restricting the bandwidth.

As you can see, this device solves some of the problems, but quite a few remain. The terminal (electronic or electromechanical) must somehow be protected from TEMPEST emanations, and needless to say, never connected to the internet. The SD card inserted into the shield must be a “clean” one. The secure SD-to-SD copier device will have to be used to copy the OTP key files and securely copy the encrypted file from this “clean” SD card to a “dirty” one which will then be inserted into, for example, an insecure PC so the encrypted file can be then e-mailed. The EAR/MIC audio ports of an insecure PC can also be used in online mode, after a G.711 link (or similar) is established between two PCs. Many LEDs are used to enable Alice to monitor the operation precisely.

The Shield is designed to mitigate many standard attacks previously analysed in this book. Using highly insecure SD cards and highly integrated MCU is not without problems. Again, the same shield could be used on a specially designed Zilog Z80 motherboard (yet to be worked out), thus evading the problems of highly integrated ICs, already discussed.

24 Frequency Shift Keying: frequency modulation used to transfer digital data (a stream of ones and zeros) through an analogue channel (e.g. analogue landline telephone line). Pulses in two distinct frequencies (1300Hz and 2100Hz here) are used to transmit digital 1 or 0.

25 Electromechanical typewriter (historical, now „obsolete“) type has less electrical but more acoustic TEMPEST emanations. Designing a TEMPEST-proof electronic “dumb” terminal (only a VDU, a keyboard and a serial comms port, without much computing power) suitable for a secure crypto system is actually very difficult.

4.4.1 • Key distribution problems

OTP is a symmetrical crypto system, so the key files (OTP random numbers sequences) will have to be somehow **securely copied** (more about this in chapter 5) and **securely distributed** between Alice, Bob, and possibly other members of their spy ring.

1. **Hand-to-hand method:** The most reliable method, but sometimes impractical and even dangerous method, especially if Eve, Mallory, or Walter are on high alert. It may be difficult to travel to meet in person, especially if Alice and Bob operate in different countries, during Cold War or Covid-19 (or its future versions) crisis. On the other hand, exchanging a few GB of OTP key files on one occasion may be enough for the next 50 years of secret communications, so they don't have to meet very often nowadays (it was more difficult during the Cold War when paper OTPs had to be carried around). Meeting in person (or a so-called "**live drop**" method) can be avoided by using a "**dead drop**", where Alice leaves the SD card in a secret place, to be picked up by Bob later. Covid-19 has made live drops today easier than ever, since wearing **full-face masks in public** is now considered **normal behaviour**, even inside a bank.
2. **Using a tamper-resistant²⁶ box** for land-mailing the SD cards. A strictly **tamper-proof²⁷** box is practically impossible to make in this instance. Eve wants to open the box and read the SD card and this is very difficult to prevent. Some designs were proposed during the Cold War using small explosive or incendiary charges to destroy the contents of the box in case of force-opening. Although secret data on paper, analogue film, or cassette tape can be reliably destroyed with an explosion or fire, the destruction may easily fail with an SD card. I don't know if this method (definitely not a discreet and even less practical one) has ever been used in practice.

The **tamper-evident²⁸** method is more likely to work. The electronic circuit inside the box needs only to reliably detect the tampering, and then alert Bob somehow. The SD card can be read, but Bob won't use it for encryption, so the OTP random numbers read will then be of no use to Eve. This box will be fully described in section 4.5.

3. **Encrypting a one-time session key** (e.g. for AES encryption) using a public key method (e.g. RSA) with Bob's public key, and then sending it to Bob is a first-rate, widely used method. This method is as strong as the RSA or AES method/key involved. It also makes no sense for OTP encryption, but it's worth mentioning.
4. **Sending an encrypted web link:** Alice first prepares a big OTP key file and then uploads it to a file-sharing web service (e.g. <https://www.4shared.com/>), then she encrypts the download link and sends it to Bob. This is a very insecure method, which may only work if Alice and Bob can access the internet from terminals that can't be traced back to them (nowadays very difficult, used to be possible in the old days, for example in an internet cafe without video-surveillance), so they don't alert Eve.

26 tamper-resistant: difficult to tamper with.

27 tamper-proof: impossible to tamper with.

28 tamper-evident: doesn't prevent tampering, but only detects it.

One unknown IP address uploaded a long compressed file, and then some other IP address downloaded it. Just a usual everyday activity on a general-purpose file-sharing website. Maybe it won't alert Eve... It may also work if Alice needs to transfer a large file to Bob, but they only need to keep it secret for a short time.

- 5. Quasar-OTP:** Quasars are quasi-stellar celestial objects, known also as quasi-stellar radio sources. They emit RF energy in a wide radio-frequency spectrum. Around 1500 quasars are known and mapped by astronomers. If Alice's and Bob's radio-telescopes are aimed at the same quasar, they can use the quasar's radio transmissions to generate one-time pads. This appears attractive at first because they don't need to meet or mail OTPs. Some quasars can be "listened to" using a microwave dish-antenna of a relatively small diameter, without a need for a big radio telescope. This method may work but under certain assumptions. The first is that quasar RF transmissions are **truly random**. This hasn't been fully proven yet, as far as I know. Alice and Bob need to tune to the same narrow frequency band - microwave, chosen somewhere between 1GHz and 15GHz range. This is the RF range used for all standard satellite-to-Earth communications. Lower RF frequencies like to refract, reflect and scatter through the ionosphere, so it is unlikely Alice and Bob could catch the same signal. Lower RF frequencies will also require reflector dishes and antenna elements of impractically large dimensions.

If tuned to the same RF frequency span at approximately the same time with the same one-time pad bit-capture algorithm, they will be able to capture the same one-time pads, to be used for secure encryption later. Considering the GHz frequencies involved, they need to listen and record the quasar signals only for only a few minutes to generate gigabytes of random numbers. The time delay between Alice's and Bob's radio receivers' reception can be a maximum of 20-40ms, considering the possible distances on Earth. This can be easily compensated. Even if Eve knows which particular quasar they are listening to, she can't know which several minutes of quasar transmission they will use every year to re-generate their one-time pads. She can't know which RF frequency span (defined by centre RF frequency and width of left and right side-bands, and also the type of the band-pass filter that they use) and which demodulation method they use to generate their one-time pads.

The next assumption is that Alice and Bob can extract the same random bit stream from two radio receivers located on two different terrestrial locations. We will see later in section 7.1. (SIGSALY-2) that an analogue encryption method (using recorded random noise as a key), is in principle similar to OTP and can be used which can make useful plaintext (or voice) legible enough above noise floor even if encryption-noise and decryption-noise "keys" are not 100% identical. The method uses a quick algorithm to synchronise both key noise signals in amplitude and phase. This is similar to searching through a one-time pad in case a key pointer is lost.

Quasar-OTP is a hypothetical method that hasn't been tested in practice yet (as far as I know) but appears to be a nice theoretical setup that could be used to design a working crypto-system if all problems are solved. Please note this is not only a method

of **distributing** but is also a method of **generating** one-time pads. Such RNGs don't fulfil all conditions for good crypto listed in chapter 3 but may be worth a try.

Extracting an OTP key from a digital data stream received from the same TV broadcast satellite (or even better, from any communication satellite in Earth's orbit!), may be an even better method. If it is a well-encrypted and compressed data stream, this is good, because then it must possess good pseudo-random properties and doesn't have to be decrypted at all, only **passively** received and securely stored to be used as an OTP key later. Passive and discreet reception of RF signal from any satellite is always possible, regardless of encryption used. There are many more active communication satellites to choose from than quasars. Their signals can be received with much simpler, more inconspicuous, and also less expensive equipment, i.e. a plain TV satellite dish.

4.5 • Tamper-evident Box solves some problems, but...

This box is designed to register unauthorised opening by erasing secret codes in an ATmega328P's SRAM. If Eve doesn't know the secret codes, she can't revert everything to its original condition and send the box on its way after reading the SD card. Bob will notice the box has been opened, so he won't use the OTP random numbers on the SD card for secure communications. The contents copied from the SD card (random numbers) will then be of no use to Eve. Seems simple, but there are many problems to be solved. Follow the link [26] for all details about the project.

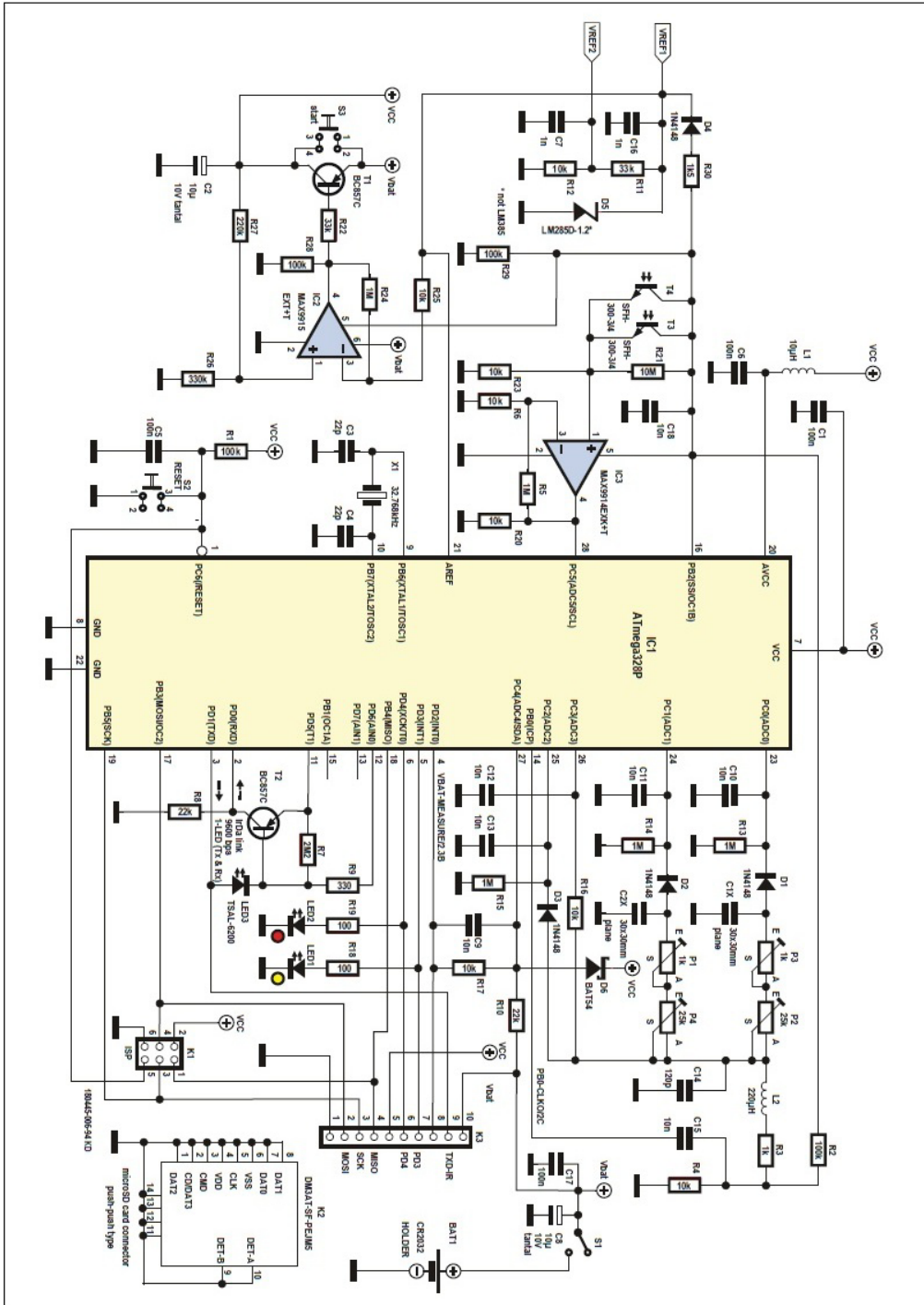


Figure 4.4 - Tamper-Evident Box schematics

The Box uses an IR LED3 as 9600bps IrDa UART (the same LED for Rx and Tx) for communication with the outside world. It is inside an aluminium box, with just one hole for LED3. This is how the box reports its status to Alice and Bob, and receives the codes (the challenge and response strings) for locking and unlocking. The IrDA communication adapter (IrDA to UART, RS232, or Ethernet terminal) can also be assembled on a DIY basis, following the instructions in the project article published in Elektor - refer to this for details of the operation. Alice locks the box by defining both challenge and response strings and then mails it. Bob unlocks it (when he receives it) by entering the challenge string. If the response string is ok, he is clear to use the one-time pad stored on the SD card for secret communications.

An unauthorised opening will zeroise the SRAM containing the codes and the travel timer (check 2.5.3, the time-stamping procedures). The detection procedures are as follows:

1. Incorrect challenge string entered 5 times.
2. The capacitor voltage ratio differs too much (Cx1 and Cx2 open-plate caps).
3. Phototransistor voltage too high (T3 or T4).
4. Battery voltage too low.
5. The temperature is out of bounds (-20 °C .. +60°C).

A standard CR2032 cell can power the box for more than one month. I designed the box after considering all possible problems described in previous chapters (e.g. SRAM data remanence, cold-boot attacks, hardware Trojans, TEMPEST, etc.). Hardware Trojans can't be mitigated by porting the system to Z80 because it would be too large and require big batteries to travel for more than a month through snail-mail. There is another way: by utilising unrecoverable **analogue memory devices** described in section 1.3. along with "primitive" **hardwire programming** mentioned in 2.5.2. and 2.5.3. It will be fully presented in the following chapter.

Since the system relies on reliably zeroising the SRAM, it will have to be upgraded to fully mitigate the dangers of hardware Trojans. Alice and Bob will have to generate their own versions of the **obfuscated code**, as described in 2.5.3.

Other problems may arise if Bob lives in an oppressive regime. If Eve opens the box, and it is perfectly zeroised, Bob will know it, but the mere fact that a packet with "weird" content has been mailed to him may be enough to put him in big trouble. A solution for Bob could be to use false ID documents and pick up the packet as a "POSTE RESTANTE" mail in the post office. A short-range UHF transceiver (in listen-before-transmit mode) would have to be installed in the box instead of the IrDA interface, so Bob can remotely check its status, from a distance of at least 100m from the post office depot. If the box has been opened, or if it simply doesn't respond on the radio link, then he won't walk into the post office to pick up the compromised packet.

This concludes chapter 4. As we have seen, it is possible to make a secure DIY crypto device, without any special components, and without the need for 3-letter agency annual government funding to do the job. We have also identified many other problems that can and will emerge. In the next chapter, we will proceed to several more fully complete and tested gadgets (most of them not published in the Elektor magazine, at the time of writing this text) that will surely fill in the remaining security gaps.

The NATO Bloc was occasionally short on multi-lingual personnel. Since the radio transmitter location was tri-angulated to East Germany, the received encrypted message was naturally forwarded to a German-speaking cryptographer for decryption. He cracked the code successfully, but couldn't read the first sentence of the plaintext (plaintext can be recognised - if it looks like plaintext), so he tossed it into a trashcan.

The message was not in the German language. The radio-telegraph operator was in East Berlin, also a pro-Soviet Polish working for STASI. He intentionally used the old code for encryption (he had been informed already by their spy in the NATO base that this code had been cracked), with plaintext in Polish, to make sure that his comrade gets the message (along with many other recipients in the West as well). This gave him barely enough time to escape to West Berlin after completing his transmission (the Berlin Wall wasn't built yet). The message was a detailed report about the Katyn Forest.

Many Polish soldiers spent WWII on the Soviet side. Although well aware of the Molotov-Ribbentrop pact, they didn't know about the war crimes committed by communists, like the infamous Katyn Forest massacre. Many of them were members of the communist party, so fighting alongside the Soviet Union against Nazi Germany seemed logical at that time. Responsibility for Katyn Forest war crime was systematically denied by Soviet authorities and blamed on Germans until 1990.

Chapter 5 • A few more cheap and highly secure gadgets

Most of these gadgets have been developed and assembled as fully-working perfboard prototypes at the time of writing this book. Professional PCBs will be designed. You can do it yourself if you want to. Some tweaking with MCU firmware may be needed, but that also goes for devices in chapter 4. They can always be further improved as well. Some of them may already be published as Elektor articles with full project support (PCB Gerber files and everything else) by the time that this book is printed. The MyNOR project by Dennis Kuschel, a 1-bit computer without an integrated CPU is already fully complete.

5.1 • SD card-to-SD card copier

SD cards have many security pitfalls (as established in 1.3 and confirmed throughout this book). On the other hand, they are difficult to fully abandon because of their ubiquity, small dimensions, low price, high memory capacity, and simplicity of interfacing. If their security pitfalls are somehow mitigated and strict security procedures followed, they can still be used in secure cryptosystems.

The TRNG, OTP Crypto Shield, and Tamper-evident Box are all designed to operate with SD cards. The copier will enable the following:

1. Since it can access every single sector of an SD card (512 bytes long), it can reliably securely copy a file from one SD card to another, without uncontrolled leakage of data between cards or from/to anywhere else. A general-purpose PC can copy an SD card, but you can't control access to individual sectors, and some data will always leak to/from the PC unnoticed.
2. It can make copies of OTP key files (containing random numbers generated by the TRNG) for Alice and Bob. It can also copy files between **clean** and **dirty** SD cards (explained in 2.6.5) while making sure the clean **SD card stays clean**, and a dirty SD card can then be used with an insecure system.
3. It can create and edit textual files, to prepare the messages for offline encryption using the OTP Crypto Shield. Other basic FAT32 file handling functions like deleting, renaming, and formatting are also possible.

Secure SD-to-SD card copier

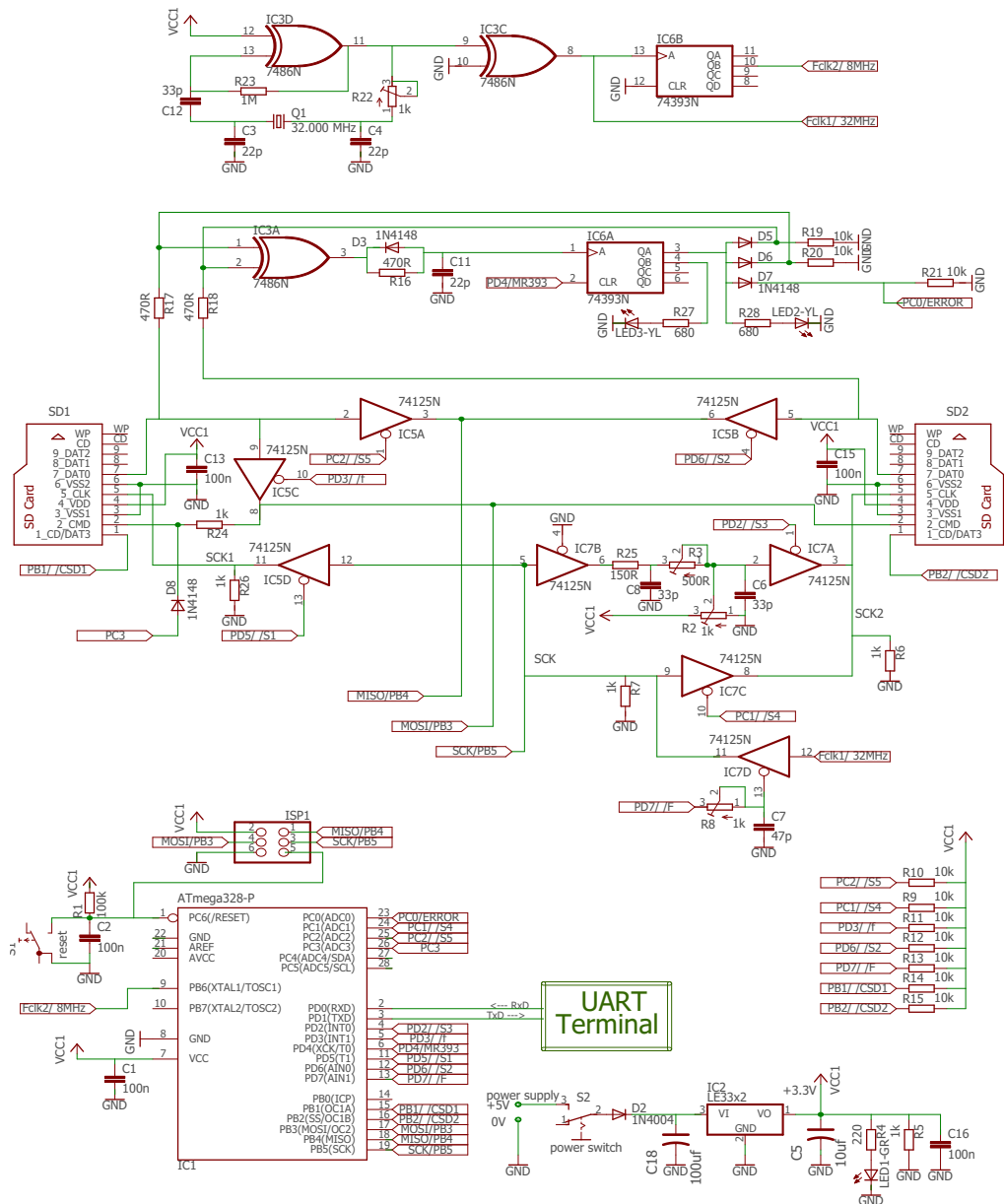


Figure 5.1 - SD to-SD copier schematics

4. DMA¹ mode for fast copying enables the bitstream from SD1 to be “injected” directly to SD2 (through IC5C), without passing through the IC1 MCU, at a speed that is too high (32 MHz) for IC1 MCU to capture it on its SPI port (if a hardware Trojan is planted inside the MCU).
5. Cloning an entire SD1 to SD2 is also possible, by copying sector-wise, regardless of formatting or file systems.
6. Verification of copied data is done also in DMA mode through IC3A, the XOR gate whose output will go high if two different bits appear on its inputs. IC6A will then latch and inform the MCU about it.

IC3D and IC3C are used to generate the 32MHz clock, then IC6B divides it by 4, to generate 8MHz for the MCU clock. The MCU can communicate with SD1 and SD2 mode in a “slow” SPI mode (SCK up to 4MHz), and then reroute SPI signals to “fast” SPI DMA mode (SCK at 32 MHz) to transfer multiple sectors of files from SD1 to SD2. IC5 and IC7 are used for routing the signals, controlled by MCU. The circuit around IC7A and IC7B is used to delay the SCK clock for one 32MHz period, to synchronise the SD1 output with SD2 input in DMA copy mode. See the lab’s project page at [27] for all the details.

Although hardware Trojans planted inside the IC1 MCU can’t do much damage (the copier usually stays inside Alice’s or Bob’s safe house, and normally never gets into a third person’s hands, along with TRNG which seldom leaves the safe house) as on OTP Crypto Shield or Tamper-evident Box, the whole system can also be modified to work with a Z80 instead of ATmega328P. The Z80 is slower (4MHz- Qc on IC6B), but the DMA transfer speed will remain at 32MHz.

With a little extra circuitry, it can be upgraded to **count pulses** on SPI signal lines, and compare them with copying files of the same size on a reference ATM328P **golden chip**. A higher number of pulses counted indicates a possible hardware Trojan activity. You need to clock the SPI with more pulses to leak extra data to SD cards.

Another, relatively complex procedure to enhance the SD card security is tweaking with its internal MCU which handles LBA addressing and wear-levelling among physical memory locations i.e. physical sectors of SD cards.

5.2 • SD card-to-Cassette tape copier

Cassette tapes are better than SD cards in every security-related aspect. This was already analysed in section 1.3. Special variants of digital tape systems designed for storing digital signals directly on a tape may be expensive, difficult to obtain for Alice and Bob, and may contain extra digital electronics that can’t be trusted (the same goes also for newer models

¹ DMA: **D**irect **M**emory **A**ccess. Transfer of data from one memory (or I/O device) to another directly, without passing the data through CPU’s internal registers and data bus. Works much faster and also relieves the load from the CPU, but requires extra circuitry. This is exactly how the SD-to-SD copier is designed to operate.

of VCRs -remember?). Above all, too many different digital tape formats are in circulation, and anybody buying this kind of equipment these days will raise a flag for Eve.

Plain analogue audio cassettes are still ubiquitous and very easy to buy. Simple analogue tape decks can't hide any extra circuits or trojans. This is why the best solution for Alice and Bob is to build this device, which will convert digital bits to analogue audio pulses, which can be recorded to a standard cassette tape at 1200bps. Furthermore, this device will work with **any audio cassette tape deck without modding (electronic or mechanical) on the deck itself**. In the 1980s the Sinclair ZX Spectrum was designed to work with any standard cassette tape deck, while the Commodore 64 used a specially designed tape deck (which on the other hand worked with standard audio cassettes).

A standard **compact audio cassette** tape runs at the speed of 4.75 cm/s reaching a useful bandwidth of 12kHz (very high-end tape decks, with state-of-art analogue corrective filters and tape speed control systems, were also built, reaching 20kHz bandwidth with standard cassettes, like Japanese Nakamichi). An audio frequency span of only 180Hz-3000Hz is needed for this device to work. This is why it can work with a dictaphone **microcassette** running at only 2.4cm/s, and even 1.2cm/s. The upper corner frequency increases approximately proportionally to tape speed. Please note that analogue telephone line modems have been built for speeds of 28.8kbps (reaching even 56kbps with digital compression) - the audio bandwidth of analogue phone lines is also around 3000-4000Hz. I am mentioning this now because it is always important to remember not **to push** the available technology to its limits when designing secure systems. Book [28] explains the operation of tape decks in detail, while [29] also covers many other retro technologies possibly interesting to Alice and Bob.

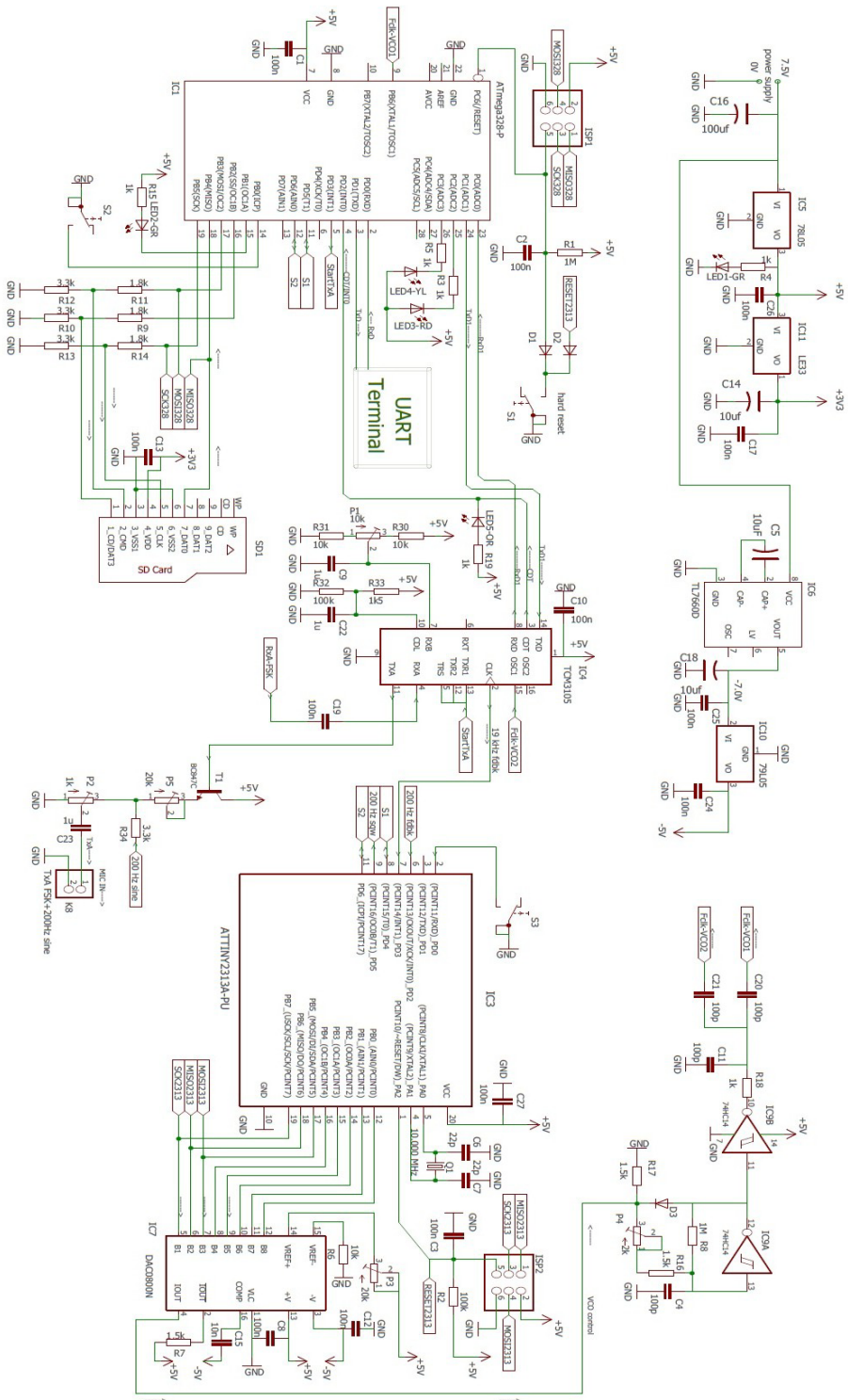
Tape speed fluctuations, called **wow and flutter** (**wow**: slow variations, below 5Hz, **flutter**: fast variations, up to 100Hz) are the main problems for data storage using tape technology. They cause tone frequency variations, which is a big problem for an FSK modem, and also amplitude variations (less of a problem). They are an even bigger problem in hi-fi music audio because sensitive audiophile's ears can notice a speed variation of only 0.5%. High-end tape decks can have even less than 0.1% variation. Reduction of wow and flutter is very demanding and requires high precision electronic and mechanical design engineering. I wanted to make this device operational with very low-quality tape decks (and cassettes as well), with wow and flutter **speed variations up to 10%**, without any modification to the tape deck itself, so I took a different approach. Instead of stabilising tape speed, I made a circuit that automatically adjusts the sampling frequency and FSK modem clock frequency, along with the main MCU clock, according to the tape speed fluctuations.

The TCM3105 modem, already known from the OTP Crypto Shield, is again used here. The frequency span required for its FSK signal is 700-2700Hz. A constant 200Hz sinewave tone is recorded along with FSK when recording data to the tape. When reading data from the tape (tape in playback, audio signal coming on EAR-headphone output), these two signals are separated using filters in figure 5.3. A circuit around IC1 is a narrow-bandpass filter, passing a span between 180Hz and 220Hz. The same filter is used also to filter the 200Hz square wave to pure sine when recording the data to the tape. The filter around IC2 is

a notch type, suppressing the 200Hz, leaving only a 700-2700Hz FSK signal. The circuit around IC3 is **automatic gain control (AGC)**, used to stabilise the amplitude variations in FSK signal (using D3 and D4 as variable current-controlled resistors) before passing the FSK signal to TCM3105 for demodulation.

The schematics of the main part are shown in figure 5.2. The pre-recorded 200Hz signal reads between 180Hz and 220Hz when playing back with 10% wow and flutter. After filtering with the narrow band-pass filter, this signal is passed to the PD2/INT0 input of IC3 (ATTiny2313MCU). The IC3 is clocked with 10.000MHz XTAL, so it can measure the frequency of 200Hz (actually 180 to 220Hz) feedback precisely. The only job for the IC3 is to constantly measure the 200Hz feedback frequency (proportional to actual tape speed) and **adjust the clock frequency for IC1** (ATmega328P-the main MCU, handling the data transfer between SD card and the tape, following the commands from the Terminal connected to its UART port) **and IC4** (the TCM3105 FSK modem) accordingly, between 4.0MHz and 4.9MHz. This way, peak frequencies for IC4 internal digital filters are varied, along with the data sampling rate (both for IC4 and IC1), so the tape speed variations are fully compensated. After measuring the 200Hz feedback frequency, IC3 sends an 8-bit word reference to IC7 (D/A converter), which then acts to increase or decrease the frequency of the variable frequency oscillator built around the IC9. This is a CPU clock for the IC4 and IC1. When communicating with the terminal through UART, the IC1 first sends a request to the IC3 to set a fixed clock frequency. The variable frequency of the SPI bus (between IC1 and SD card) is not a problem. If you decide to build it DIY refer to labs page [30].

Secure SD-cassette copier



Secure SD-cassette copier, analog part

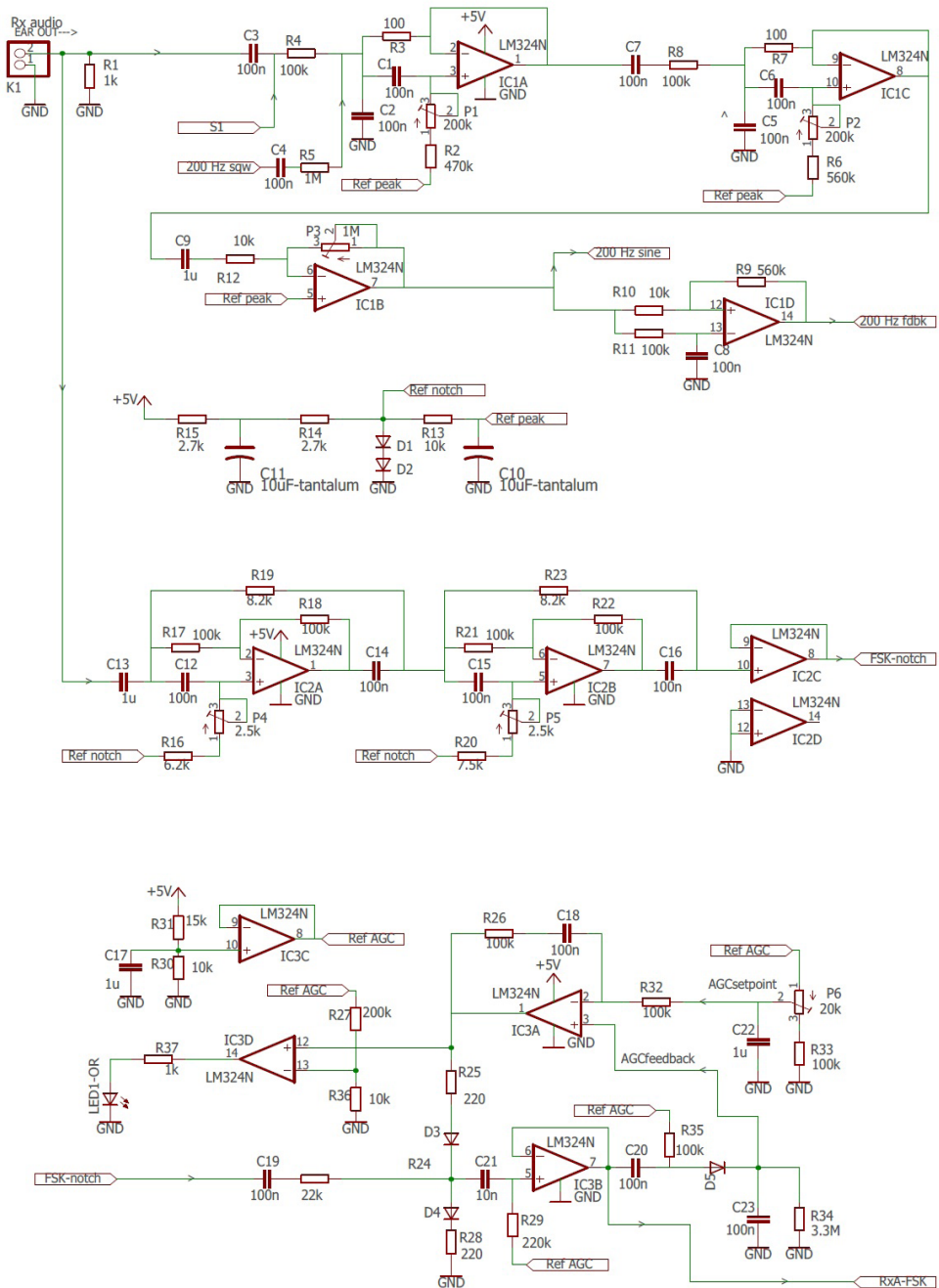


Figure 5.3 - SD card-to-cassette tape copier schematics, the analogue part



Figure 5.4 - A compact cassette and a microcassette tape

Cassettes with encrypted messages and files can be land mailed (in a plain envelope!) between Alice and Bob because cassettes can be trusted not to contain any unwanted secret data, e.g. “leaked” plaintext, or anything else a Trojan can secretly copy to some secret sector of an SD card. A microcassette with OTP key files can easily fit inside the tamper-evident box to be securely mailed, instead of an SD card, if needed, for extra security.

5.3 • ZMC80 system by Lee Alan Hart

The possible use of the Zilog Z80 system for secure cryptography has been proposed many times in this book already. The first system I considered was Sinclair ZX Spectrum, which I quickly rejected because of the integrated UHF PAL video transmitter. The next was “**Galaksija**” by **Voja Antonić**, a very simple and cheap Z80 computer built at the beginning of the 1980s, designed to be possible to be assembled inside Yugoslavia with all legally obtainable parts, without a need for any smuggling (like e.g. ZX Spectrum). It featured a Z80A bit banging a 1-bit colour picture to a TV, leaving some 20% of its CPU time to do some useful work. While trying to find it on the internet, I stumbled upon **Lee Alan Hart’s remake** of Voja’s Galaksija [31], with a composite video output (no RF modulator).

I liked it, but then found that Lee had already designed an even better system, the **"ZMC-Z80"**, which is Arduino-size, and very simple. It is good for learning to program the Z80 at the lowest machine code level (like Galaksija). It also has a simple BASIC interpreter (again like Galaksija and Speccy) for those that might be afraid to start off with assembler (like me, in 1985), and all the detailed instructions.

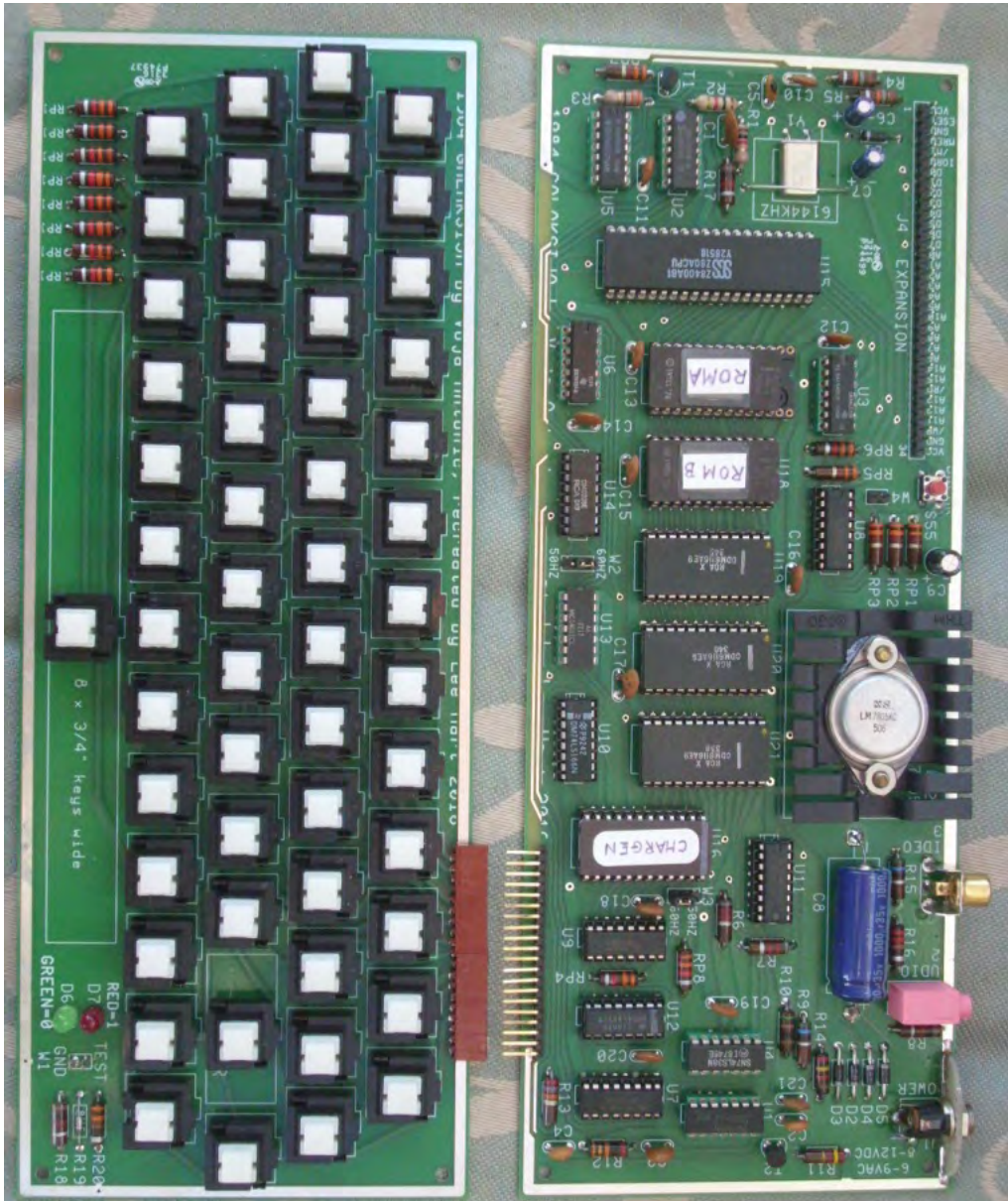


Figure 5.5 - "Galaksija": a remarkable piece of ex-Yugoslav electronic engineering history

Unlike Galaksija (featuring a keyboard and analogue video output), the ZMC Z80 system needs a **dumb terminal** (a keyboard, VDU, and serial port). We will address TEMPEST-proof terminals later, in chapter 7 (also required for almost all secure crypto hardware that I designed). Lee designed several add-on boards to the main Z80 motherboard, a front panel board (FP) with a simple 7-character-7-segment LED display, buzzer, serial port and keypad, and one SD-card/serial port interface board with a **bank-switchable** RAM (explained in 2.6.3, the Z80 can address only 64KB of memory directly with its 16-bit wide address bus). It can even run a CP/M operating system in this configuration!

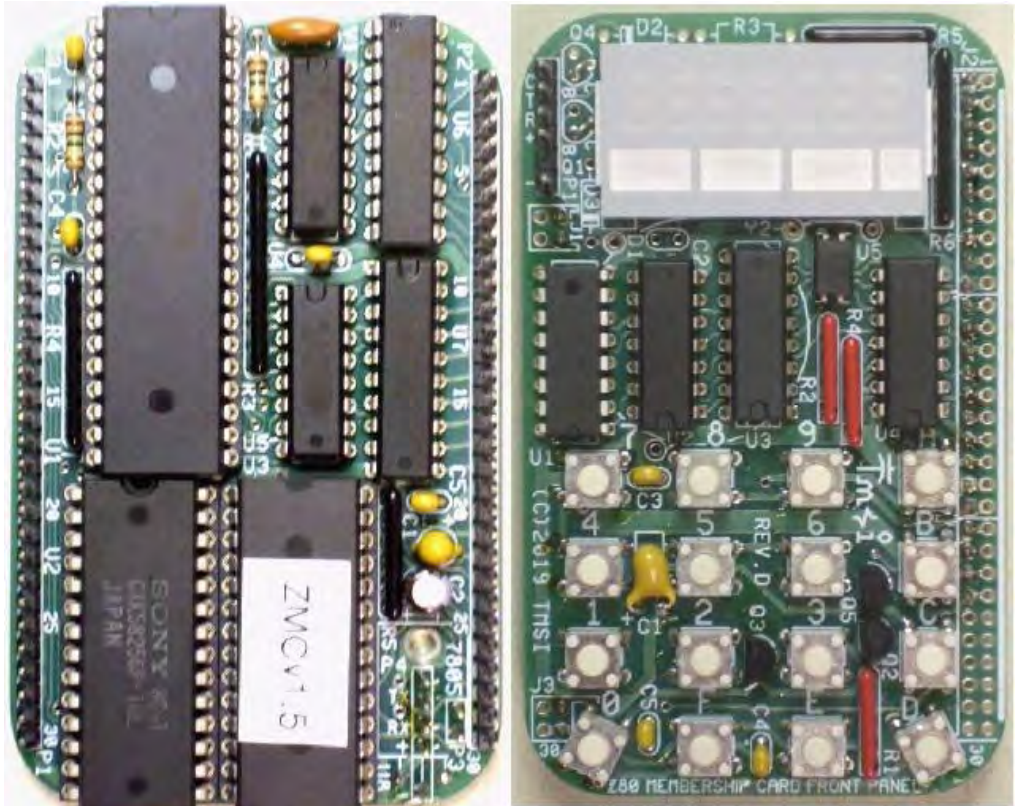


Figure 5.6 - Two main components of ZMC Z80 by Lee Hart - the main and FP PCBs

The ZMC motherboard features a bare minimum Z80 computer that is very versatile and expandable. Expansion boards are stackable, similar to Arduino. Figure 5.7 shows the schematics of the mainboard. A standard 27C256 32KB UV EPROM is used as ROM (memory address range 0x0000-0x7FFF).

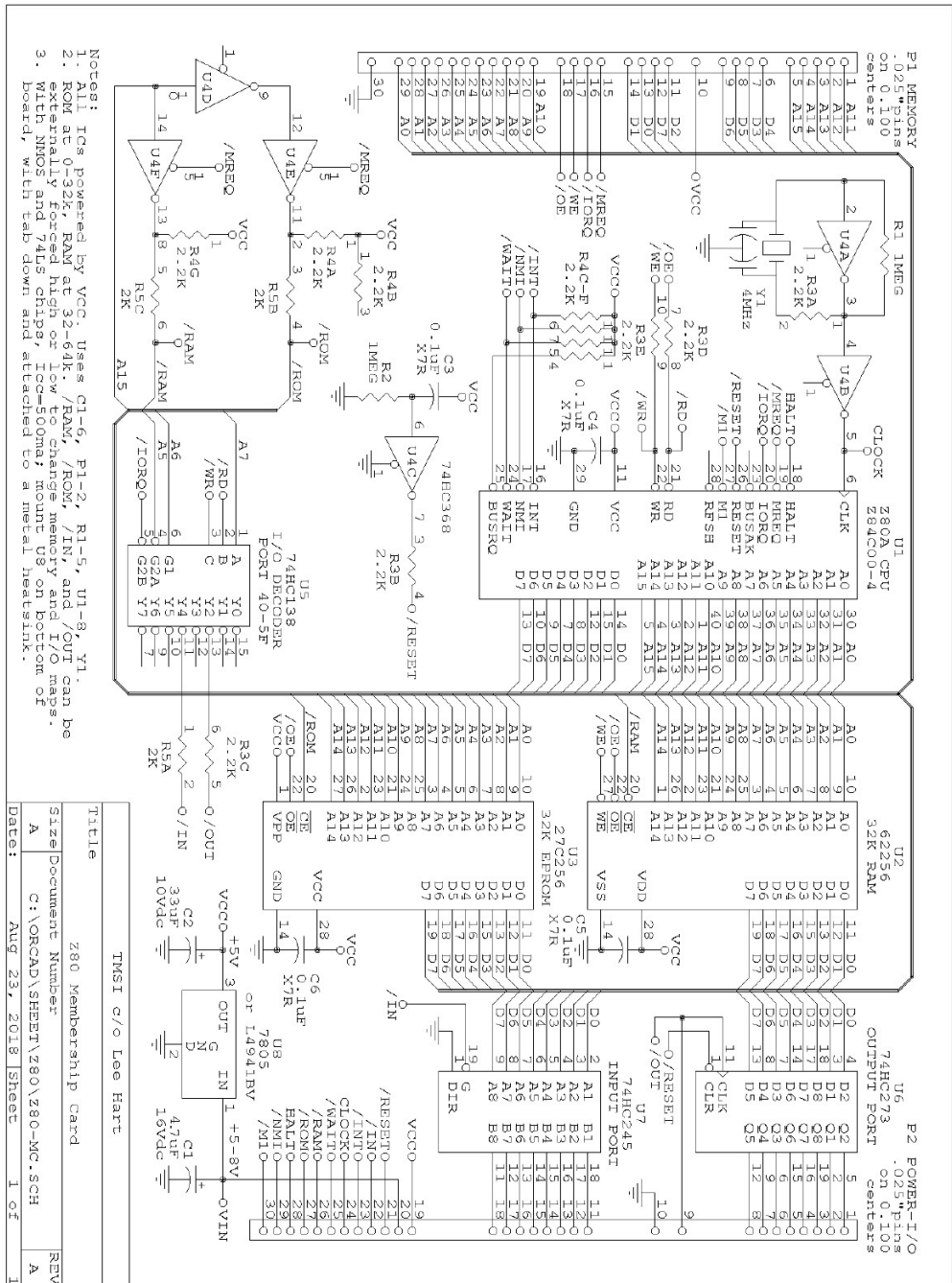


Figure 5.7 - Schematics of the main ZMC Z80 board

A 62256 32KB SRAM chip serves as RAM memory (address range 0x8000-0xFFFF). Here is also a 4MHz clock oscillator, reset circuit, 8-bit input (U7), 8-bit output port (U6), and an I/O port decoder (U5). Although the Z80 can address 0x00-0xFF (with eight lower Z80 address bus bits A0-A7) 256 I/O units altogether (which is never needed), only three address bits (A5-A7) are used at the I/O decoder (U5-74HC138) for the sake of hardware simplicity, meaning any I/O address between 0x40 and 0x5F will activate U6 (8-bit output) or U7 (8-bit input), depending on /WR or /RD output selection (write to output or read from input). This used to cause conflicts (in the old days) between different peripheral units connected simultaneously to the ZX Spectrum (two units could get activated and try to assert the data bus at the same time with one I/O address) if they were not from the same manufacturer when their I/O address ranges started to overlap.

When FP is installed, the ROM firmware required to drive the LED display and buzzer, scan the keypad, and even perform **machine-code monitoring**² along with **single-stepping**³ function will consume only 12% of CPU time, leaving 88% free for the development of user programs (like crypto-software). All necessary manuals can be downloaded from [31], containing detailed schematics, memory maps, I/O ranges used by Lee Hart's devices, description of operations, and everything else. PCBs and full kits of parts for DIY assembling can be bought here as well.

5.3.1 • Crypto development shield add-on

After seeing all this, I quickly decided to make a shield for this system to develop highly secure crypto devices. This would be more secure than those based on modern Atmel AVR MCUs. Check [32]: there you will find all details, with demo screenshots and videos. With 64 KB of ROM and RAM and flexible von Neumann's architecture, this Z80 system has more than enough hardware resources - certainly more than an average 8-bit MCU (except for speed).

2 machine-code monitor: a program subroutine used to display CPU registers and memory contents simultaneously while running the main program.

3 single-stepping: executing only one single assembler instruction at a press of a button. Very useful for debugging, but also for learning machine code at the lowest level. Learning it is a must, if you want to develop good crypto systems. It is easier to make a single-stepper for executing instructions from RAM than from ROM, but Lee Hart's unique single-stepper (controlled by FP timer-interrupts on /INT, activated at each single-stepped instruction) does ROM instructions as well.

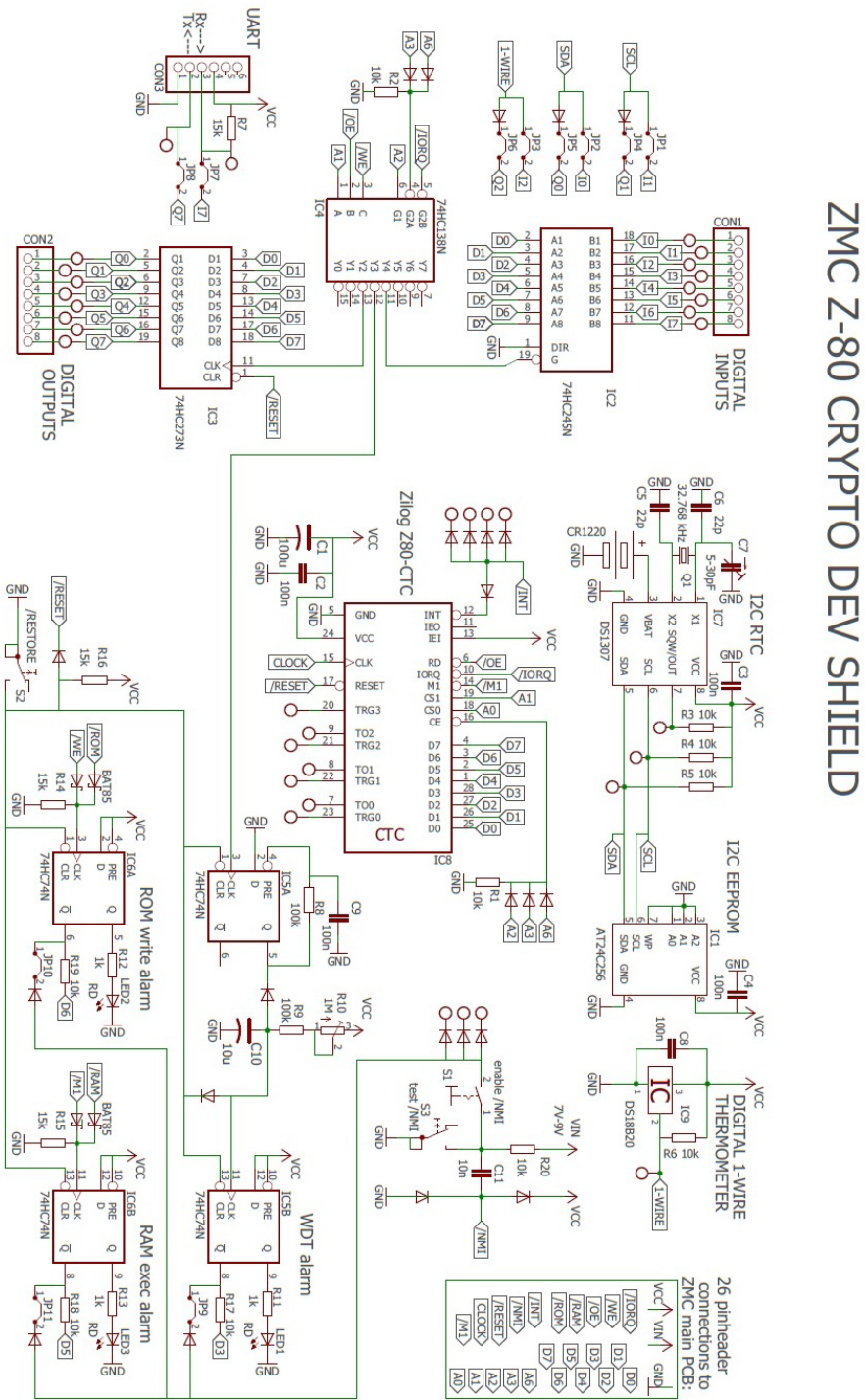


Figure 5.8 - Schematics of the ZMC Z80 Crypto Development Shield

The security pitfalls of von Neumann's architecture can be mitigated with extra circuitry included on the shield, which will (on hardware level!) prevent buffer-overflow attacks.

IC2 and IC3 (on the schematics in figure 5.8) provide an extra 8-bit input and 8-bit output port. The I/O port assigned to them is 0x04 (read request activates IC2, write request activates IC3). The circuit also features an extra UART port, one I²C port, and one "**1-wire**"⁴ port. They are all bit banded by software, using pins on IC2 and IC3. Jumpers JP1-JP8 can be removed for communication ports that are not used. IC1 (AT24C256 - 32KB EEPROM memory) and IC7 (DS1307 **-RTC-** a real-time clock timekeeper module with CR1220 battery) are connected to the I²C port. Other I²C devices can be connected in parallel if required. IC9 is DS18B20, a standard digital thermometer (to detect abnormal temperatures, like in Tamper-evident Box), read through a 1-wire port. More 1-wire devices can also be connected.

IC8 is a Z80-CTC (counter-timer circuit) that can perform various pulse counting and timing tasks and generate /INT interrupts. It has 4 independent counter/timer channels (more than many modern 8-bit MCUs), CPU I/O channels 0x00-0x03 are assigned to them. I tested it also with UB857D (East German clone) at 4MHz and it worked OK. Small circles on schematics (like those at IC8 TO0-TO2 and TRG0-TRG3 pins, at I²C and 1-wire lines, or around /INT and /NMI triggering diodes) are SIP sockets, where wires used for **hardwire programming** can be inserted. Hardwire "programmed" settings can't be read by Trojans. Z80 CTC can be used (among other tasks) to do the timing for UART (connection at CON3), I²C, and 1-wire ports.

Besides the /INT hardware interrupt input, the Z80 has another: /NMI -this stands for **non-maskable interrupt**. /INT interrupt can be enabled/disabled (masked) by EI and DI assembler commands. Unlike /INT, the /NMI interrupt will **always** react, by jumping to the 0x0066 ROM memory address. This is another good feature of the Z80 for use in secure crypto devices! If a code-injection attack is successful, it will immediately disable /INT interrupt (DI command), but can never disable the /NMI interrupt. This way, the /NMI input will always react if abnormal activities are detected, and the /NMI interrupt routine will take over.

4 The "1-wire" communication protocol was introduced by Dallas Semiconductor. It uses only one wire for asynchronous Rx and Tx, with multiple slave devices on the same line if needed. The same wire can also be used for powering the slaves. A 1nf capacitor is used at every slave to store charge for power while communicating with the master.

5.3.2 • Buffer-overflow protection on hardware level

If enabled by switch S1, the /NMI can be activated in three different ways by IC5 and IC6 (other /NMI triggers can be “programmed” by wires as well). A **watchdog timer** (WDT⁵) is implemented with IC5A and IC5B. The setting of R10 trimmer defines a time constant of up to 7 seconds. If the WDT is not periodically reset in time by discharging C10, (which is done by writing to I/O port 0x06 and thus triggering the IC5A) the IC5B will latch and activate /NMI if enabled with JP9. Please note that a WDT implemented with a special circuit (like here) is a lot more secure than one integrated inside a modern MCU. This can’t be disabled using software commands because it is all **hardwired!**

In a normal secure crypto operation, the program will never be executed from RAM, only ROM. This way, an attempt to **execute an instruction fetched from RAM immediately** indicates a **code-injection attack**. When fetching an instruction from memory, the /M1 Z80 CPU output pin will go low. If /RAM line is also low (A15 high - RAM asserted), this means that an instruction is being fetched from RAM. This will latch the IC6B and activate /NMI if enabled with JP11 to prevent the execution of that instruction. The program control will immediately transfer to /NMI service routine starting at 0x0066.

Besides trying to execute an instruction from RAM, there is also one more highly suspicious activity, and this is **trying to write to ROM**. A normal cryptography program will never try to write here. Trying to write to ROM can’t do any harm by itself, but is still an indicator of abnormal activity. When executing a buffer-overflow, Mallory will probably try to inject a long sequence of 0x00 bytes - a so-called **“NOP⁶ sled”**. If she wants to reach the beginning of RAM (around 0x8000 memory address), she will start injecting NOPs at the end of RAM, then wrap around from address 0xFFFF to 0x0000, and continue through ROM until the beginning of RAM at 0x8000. If /WE line (a request to write to memory) and /ROM (A15 low- ROM memory asserted) go low at the same time, the IC6A will latch and activate /NMI through JP10.

5 WDT is an additional timer that usually uses a simple RC circuit (like here) designed to reset the main CPU, or trigger a special interrupt, like /NMI here. WDT comes integrated within any modern MCU. WDT is almost always used not only in high-security, but also in safety-critical circuits. When a normal CPU program is executed, it always resets the WDT in regular intervals. If a CPU “freezes” or remains stuck in a loop, or if a malicious code is injected, it may not reset the WDT, so it will time out and warn Alice that something strange is taking place.

6 NOP (no operation) is an assembler instruction that makes a CPU do nothing except waste a few clock cycles. It is implemented in every CPU and is almost always represented by 0x00 stored inside program memory.

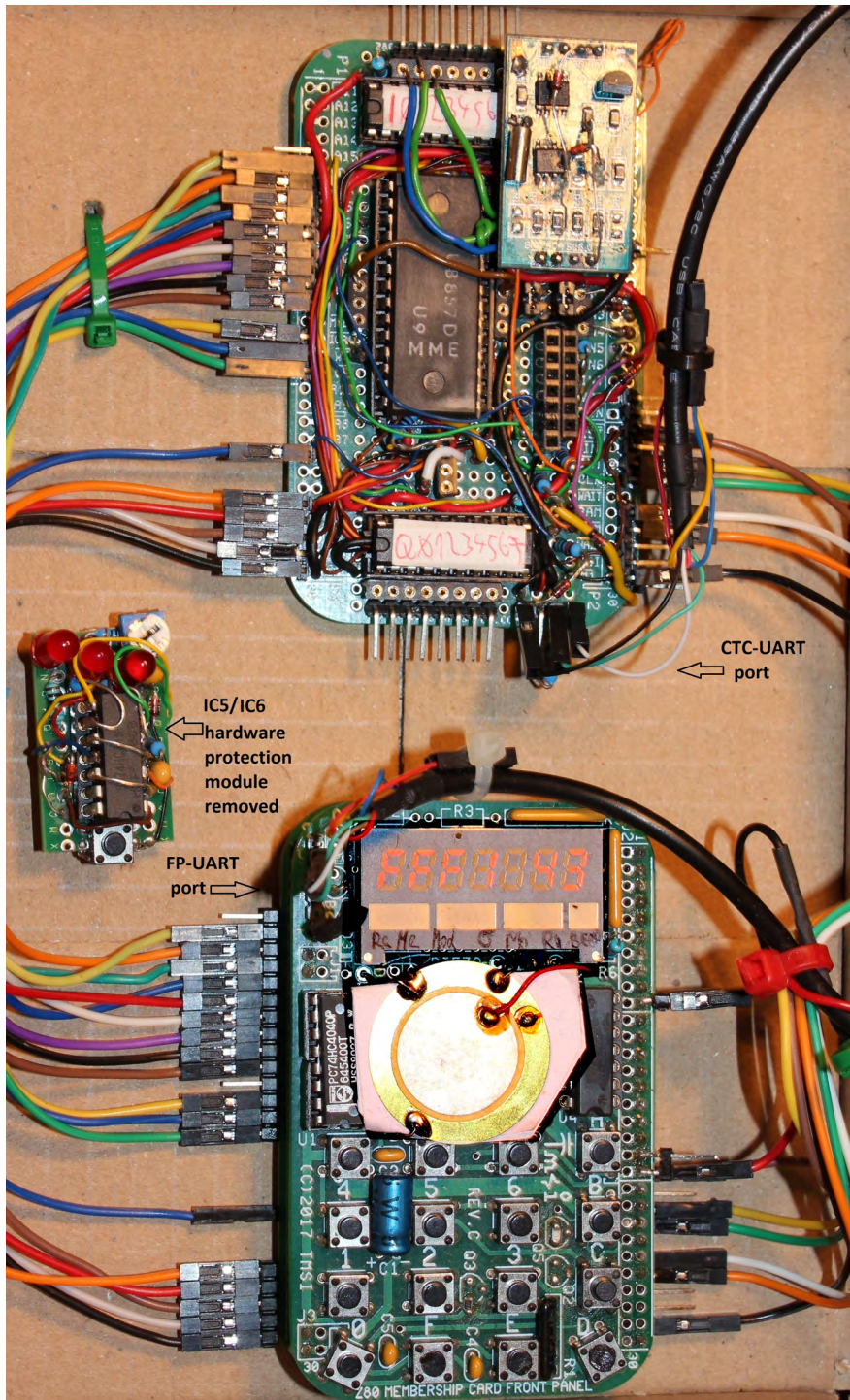


Figure 5.9 - Crypto Dev Shield on a perboard connected to a ZMC Z80 with FP panel

Any one of these three critical events will also light up a corresponding red LED (LED1, LED2, or LED3). IC5 and IC6 outputs are also connected (through resistors R17, R18, and R19) to data bus lines D3, D5, and D6. This combination of bits (indicating what caused the /NMI interrupt) can be read by the CPU by reading the I/O channel 0x06. This will activate Y5 output on IC4, which is not connected. With no I/O unit to take the data bus, the R17, R18, and R19 resistors will pull it high or low. IC5 and IC6 will always be reset along with the CPU, but they can also be reset by pushing the S2 - /RESTORE button without resetting the CPU. A circuit around R20 and C11 is used to ensure to swing the /NMI line low enough. Depending on a particular Z80 implementation (NMOS, CMOS, different manufacturers, etc.) it has sometimes failed without this circuit when testing.

This Shield was proven to work OK during tests with a perfboard prototype (see pictures and videos at [32]). The PCB still needs to be designed. Other shield upgrades for crypto applications with this system could be worked on, like ADC and DAC converters, voltage comparators, FSK modem, soft-shutdown circuits... These could all fit nicely on one more stackable PCB shield.

5.3.3 • Stack smashing and code obfuscation

These two have a lot in common, so now it's time to address this. In sub-section 4.2.1.2 we saw how it is possible for Alice to hide a piece of program code (from a hardware Trojan) inside a look-up table (normally containing constants and text strings, but not program code). On the other hand, when Mallory smashes the stack on Alice's computer, she can also exploit a part of Alice's look-up table to execute a program code. If she knows what is inside Alice's ROM, she could theoretically find a part of her look-up table containing (when translated from text to raw hex) a sequence of assembler commands that can be executed, doing something useful for Mallory. Alice probably didn't recognise them (possible sequences of commands useful to Mallory) when arranging her texts and constants inside the look-up tables, but Mallory found them later when carefully analyzing Alice's code. Mallory then maybe **doesn't need to inject any payload code** at all. It may be enough to just plant a fake return address on the stack, to jump to a certain address inside the look-up table. Alice has already prepared the "payload" code here (unknown to herself!) that can benefit Mallory! Very unintuitive, isn't it? Please note that hardware protections on the Z80 Crypto Shield won't protect Alice against this kind of attack by Mallory, if she manages to alter only one return address on the stack, without "writing" to ROM or executing commands from RAM.

There is even a more devious variant of this attack! This variant doesn't target the look-up tables inside ROM, but part of the ROM containing program code that is intended to be executed as a part of the normal operation of Alice's cryptosystem. All that is needed is the same as before: a careful analysis of Alice's assembler code by Mallory, and just one altered fake return address on the stack. Consider the following two short examples:

Address	Alice's Mnemonic	Code	Mallory's Mnemonic
0x0100	IN A, (0x01)	0xDB	
0x0101		0x01	LD BC, 0x01E6
0x0102	AND A, 0x01	0xE6	
0x0103		0x01	

Address	Alice's Mnemonic	Code	Mallory's Mnemonic
0x0200	DJNZ, 0	0x10	
0x0201		0xFE	CP A, 0xCB
0x0202	RRC, B	0xCB	
0x0203		0x08	EX AF, AF'

Check the Z80 assembly instructions - hex opcodes list and you will see if a normal return address on the stack is 0x0100 or 0x0200, a normal code programmed by Alice will be executed (in the left column). On the other hand, if Mallory alters the return addresses on the stack to 0x0101 or 0x0201, an “alternate” or “hidden” code will be executed (in the right column). Since the Z80 instructions can be 1, 2, 3, or even 4 bytes long, there are many such possibilities. Please note that Alice can also deliberately use this technique to hide her program from Trojans (Trojan sees just a sequence of hex numbers in the “Code:” column), like in the previous variant (hiding in a look-up table).

These procedures can easily lead to a “**runaway code**” condition if not performed carefully. Alice will easily notice her computer getting out of control, so she will reset/restart it. This may also be enough for Mallory to cause sufficient damage to Alice and Bob in the middle of a critical operation.

5.4 • Mg-flash analogue memory for Tamper-evident Box

I designed this add-on to the Tamper-evident Box (section 4.5) to counter the dangers of possible hardware Trojans planted inside an ATmega328P. There was no option to use a Z80 system (too big and power-hungry) for the box expected to run on a small battery for more than a month. Check [33] for all details about the project including schematics, and videos. Every digital form of memory known to mankind suffers from some sort of residual remanence, which I already talked about plenty in this book. The security of the box relies on 100% sure zeroisation of the codes and travel timer inside the SRAM. Let's suppose that MCU SRAM can be recovered and that Mallory can plant a trojan to periodically copy the SRAM to some secret flash memory space.

My basic idea to solve this problem is to use retro low-tech again. If digital can't do the job, let's go analogue! Paper, cassette tape, or analogue film tape will lose the data if overheated or even overexposed (analysed in 1.3). If Eve can't recover the data, she can't revert everything to its original condition, so Bob will be warned. This is the basic idea. The cassette tape copier (section 5.2) will be very useful here. The schematics are in figure 5.10.

In the case of tampering, a magnesium photo bulb will fire and destroy the "special" response code written on paper, cassette tape or analogue film tape. Mg-bulbs, one-shot disposable (e.g. a standard AG-1B type) use a very high temperature burning magnesium to create a bright flash. They have been used in photography for artificial illumination since the beginning, and they are still being produced nowadays for use in analogue and digital photography [34].

Besides the already mentioned planted hardware Trojans, this circuit can also ward off cold-boot attacks. Although difficult to be successful against the box, because it has low-temperature protection and erases the SRAM on each restart, they are worth mentioning. Recovering SRAM this way also won't help if the Mg bulb destroys the information on analogue media.

Three capacitors (C1, C2, and C3) hold enough energy to trigger the Mg-bulb even without a command from the MCU in case of a short circuit or low voltage on Vcc or Vbat. The circuit also features a barometric air pressure sensor (IC3-BME280), to quickly react to drilling attacks.

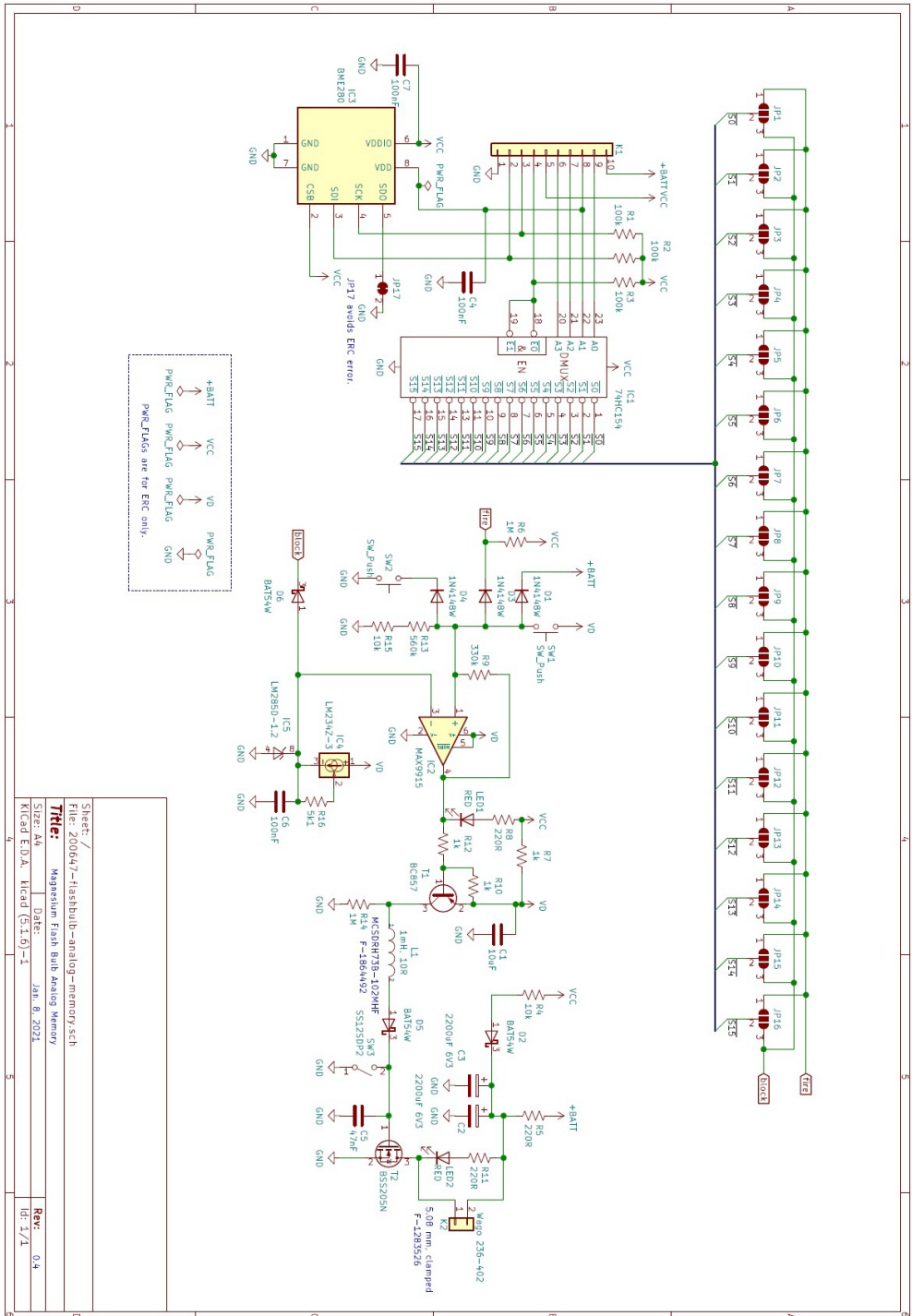


Figure 5.10 - Schematics of the analogue memory add-on to the tamper-evident Box

The K1 connector plugs into the K3 connector on the main PCB. This is why K3 was installed on the main PCB. Unlike the main MCU circuit, which wakes up from power-save mode every 2 seconds, the add-on is constantly active (to immediately react to possible short circuits or voltage drops in case of a fast drilling attack). This is why low-power ICs, for safe operation down to $V_{cc}=2.0V$ are used, for a total extra supply current of only 20-30uA. IC4 provides a constant current for IC5, a precise 1.23V wide-temperature range voltage reference.

MCU reads barometric (absolute) air pressure from IC3 every 2 seconds, using a bit-banged I²C interface through K1. IC1 is a standard 4-to-16 decoder, used to activate “fire” (to fire the Mg bulb) and “block” commands (block firing circuit, when correct code is entered). Outputs for both commands are chosen in a “hardwire” way, using solder-jumpers JP1-JP16. IC2 is wired as a comparator with positive feedback, to react to low voltage on its pin 1(+). SW1 is used to arm the trigger, and SW2 is for test-firing. C1 stores enough charge to turn on the T1 (and power IC2, IC4, and IC5), in case of a low voltage-short circuit on V_{cc} or V_{bat} . L1 and D5 will make the C5 voltage swing higher than V_d , to fully turn on the T2. C5 will ensure to keep it on for at least 10-20ms needed for Mg-bulb clamped at K2 to fire. C2 and C3 store energy needed to provide a 10ms, 1-2A current pulse for ignition of magnesium.

SW3 is a safety switch, which disables the Mg-bulb firing by holding a T2 gate at GND. It can be released when everything is ready. LED1 and LED2 are red warning lights. If one or both are on, something is not ready, so don't connect a bulb to K2 or release the SW3. When the MCU decides to zeroise the codes, it sets IC1 inputs A0-A3, /E0, and E1/ to a combination to trigger the “fire” output (D3 cathode). If a correct code has been entered to unlock the box, the IC1 inputs will be set to trigger the “block” command (D6 cathode). This lowers the voltage on the 3(-) pin of IC2 down to 0.3V, so IC2 can't fire. In this case, the MCU will stay in active mode, and the box can be opened. The SW3 safety switch can then be closed, and the Mg-bulb can be removed from K2. Bob can now unwrap the paper/cassette tape and check the “special” response code.

IC1 and JP1-JP16 work as a protection against an application-specific Trojan that may disable the “fire” command. IC1 inputs will be constantly shuffled by MCU firmware, and the firing and blocking combinations are set in a hardwire way through JP1-JP16, so Mallory's application-specific Trojan can't read them. An application-specific Trojan can't work if it can't recognise the firmware or outputs. A general-purpose Trojan can copy the codes from SRAM to a secret flash memory, but can't copy the codes from paper or cassette tape wrapped around the Mg-bulb.

When arming the box, some of the air will be pumped out (e.g. down to 0.8 bar pressure), using a vacuum pump. This way, the air pressure is registered by IC3, read by the main MCU, and compensated for temperature. This will detect a drilling attack within 2 seconds (the MCU power-save mode time). The Mg-bulb, with paper or cassette tape containing the secret special response codes wrapped around it, can be connected to K2, after the add-on PCB is plugged in the main PCB, powered up, SW1 pushed to arm the trigger and both LED1 and LED2 are off. Then SW3 can be released. The whole assembly is then put into a box which can be “locked” by codes entered through the IrDA interface.

Any type of cassette tape will be categorically burnt by an AG-1 Mg-bulb. If paper is used, a thin cigarette rolling type paper is recommended. It is also recommended to dip it in a saturated water solution of potassium permanganate and let it dry, then write the special code on it. A strong oxidizer lowers the ignition temperature of paper. None of these has failed to ignite when tested.

5.5 • Security by obscurity

Now it is time to fully address this, although it has been previously analysed in this book. It is a term describing bad security, which can be defined as relying on hidden procedures (which will become widely known sooner or later) rather than on keys to achieve security. If all program code and hardware is open-source, and thus known to anybody (including Eve and Mallory), and only keys and passwords are kept secret then it is **good security, not security by obscurity**. Good examples are ENIGMA (algorithm was publicly known, but keys kept secret - and it was still difficult to crack), PGP, AES-128, OTP, and any good well known mechanical lock.

Bad security by obscurity is widespread. An example is the Eurochip card which relies more on the secrecy of the authentication protocol, than on its strength. When testing a new mechanical lock, the time required for a good locksmith to open it **after the mechanical details of the lock are known** is what counts. If it takes 2 hours the first time, but only 5 minutes later, after the lock is disassembled and design weaknesses are found, it is security by obscurity. Hiding one-time pads somewhere on the internet, or generating them based on quasar transmissions (described in 4.4.1), is also security by obscurity up to a certain level because Eve can (theoretically) still access the keys and read them. Another example may be hiding an apartment key under a doormat, or hiding a message in a picture (steganography), without encrypting the message - both not secure enough. Using a new, secret, but not well-tested encryption method (e.g. like encryption that mobile phone operators used with GSM in its early days) is also security by obscurity. The same goes for any black-box device or highly-integrated device, where the manufacturer guarantees “security”.

Relying on **obscurity alone** to protect your secrets is a very bad security practice, but regularly used by many people. On the other hand, many methods with elements of obscurity have been presented already in this book. It is OK to use these **methods combined with good methods**. This definitely **improves security**. Steganography, code obfuscation, scrambling, dead-drops, and similar methods all rely on obscurity. They are OK to use if

combined with good encryption. In the case of the tamper-evident Box, secret codes (in SRAM and on paper) and hardware programming are combined with code obfuscation (a method that relies on obscurity) to improve overall security.

5.6 • MyNOR CPU-less computer by Dennis Kuschel

This project [35] seems like a perfect way to end this chapter. The ZMC Z80 system achieves more security by low integration, but still has a highly integrated CPU. MyNOR takes it even further: this computer has **no integrated CPU!** This way, it is even a more secure platform, because Mallory can't plant a hardware Trojan inside a CPU, simply **BECAUSE THERE IS NO CPU!**

MyNOR features a RISC architecture taken to the extreme. A NOR gate (two discreet MOSFETs and a pull-up resistor) serves as a **one-bit "ALU"**, with only **one machine instruction implemented in hardware** - a NOR (NOT-OR) between two inputs. All 8-bit operations are performed serially, as a series of 1-bit operations. All arithmetic and logic operations are therefore implemented as sequences of NOR operations - (" $+$ " means logical OR, " $\cdot$ " means AND, " $\oplus$ " means XOR, " $/$ " means NOT):

$$\begin{aligned} /A &= /(A+A), & A+B &= /(A+B), & A \cdot B &= /(/A + /B) \\ A \oplus B &= /(A + /B) + /(/A + B) \end{aligned}$$

Half-adder outputs two bits S (sum) and C (carry) from two input bits A and B:

$$S = A \oplus B, \quad C = A \cdot B$$

Full-adder outputs two bits S (sum) and C_{out} (output carry) from three input bits, taking also a carry-bit from previous addition into count:

$$S = A \oplus B \oplus C_{in}, \quad C_{out} = (A \cdot B) + (C_{in} \cdot (A \oplus B))$$

Eight full-adder operations in sequence are performed to accomplish one 8-bit addition. These sequences (for ADD and all the other commands) are controlled by microcode (takes 9KB of 32KB ROM memory), which supports 28 assembler instructions, **implemented in software-microcode**, as a series of 1-bit NOR operations.

This leads to reduced speed, but it can still perform simple calculations (like for example a one-time-pad XOR) in a reasonable amount of time. At 4 MHz it can perform up to 2600 8-bit additions per second. A Z80 can theoretically do 1.000.0000 per second (additions among CPU registers are only one-byte instructions, e.g. ADD A, B - opcode 0x80), but these registers need to be loaded, and results must be stored somewhere. It comes down to some 100.000 useful 8-bit additions per second. The PCB is small and simple (much smaller than what you would expect!). All documentation is available at [35].

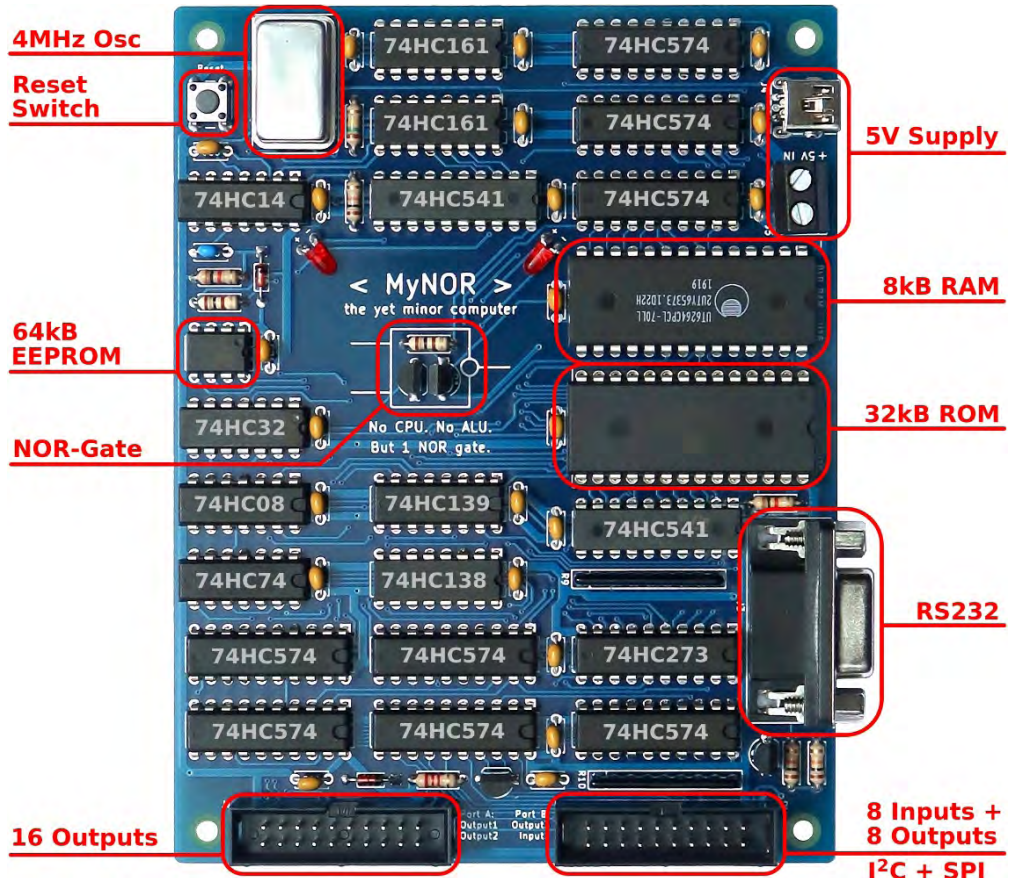


Figure 5.11 - MyNOR PCB is much simpler than what you might expect!

Please note the 27C256 32KB EPROM in **OTP** (“OTP” can have a different meaning, here it stands for **one-time programmable**, but it can also be a **one-time password** or **one-time pad**) variant. This is a cheaper variant, without a transparent UV lens, so can be programmed only once. There is no possibility of UV light leakage, so this variant has a better, more reliable long-term retention of data. It is, therefore, better to use this than a UV-erasable variant, if the code is not to be changed. There is even less opportunity for Mallory to change a part of the code here than (theoretically) with the UV version.

Alice and Bob can take it even one step further and use old OTP **PROM** memory. This type of non-volatile memory is programmed by **burning nichrome fuses inside a chip**. Burnt fuses (each burnt fuse stands for one **digital zero bit** programmed-saved to PROM memory, and each fuse left intact stands for one **digital one bit**) can’t be restored, so it can be programmed only once. Its long-term data retention is much better than UV or OTP EPROM.

I chose this project to end chapter 5 to show that it is always possible to take a high-security design one step further.

A one-bit ALU, CPUless computer, OTP EPROM, PROM - you probably haven't thought of any of this before reading this final section of this chapter.

At the time of writing this book, Dennis Kuschel is also working on **TraNOR**, a computer fully compatible with **MyNOR**, but implemented with some **2500 discreet transistors**. All 74HC chips from figure 5.11 were **removed** from a MyNOR board, leaving only 8KB RAM, 32KB OTP EPROM, and 64KB EEPROM chips. I'd like to see Mallory plant a Trojan on that system...

The next chapter will give you an idea about how to perform practical attacks against simple systems, using straightforward and cheap hardware. No black magic was involved, only the basic principles learned in the previous chapters. It's time to test them in "real" action!

An embassy received an anonymous warning about a microwave oven purchased for its kitchen. The year was 2000. The warning was about a covert listening device planted inside the oven. The oven was taken apart and checked in detail by their technician, but he didn't find anything suspicious, so the oven was allowed to be installed in the kitchen.

Needless to say, a lot of critical info leaked from the embassy's kitchen in the following months, and one high-level embassy official died while operating the oven, without raising much suspicion.

How come?

Chapter 6 • Hands-on!

In this chapter, we will demonstrate a few practical attacks. The theory previously explained will now be put into practice. The hardware used is cheap and simple. You can easily try repeating the experiments. First, we will attempt two TEMPEST attacks (explained in 2.4.1). After that two buffer-overflow attacks (section 2.3.1) on a Z80 system (von Neumann CPU) and two memory-remanence attacks (section 2.6) on a standard SRAM. If you read and understood the theory, you will probably think of your own variants, or ways to enhance and automate what's shown here. **Do try this at home**, please!

6.1 • TEMPEST attack demos

In most cases, Eve must start with some intelligence work. She needs to gather technical information about Alice's hardware she is trying to attack. Even if Alice works for a well-funded 3-letter agency, she probably uses a general-purpose keyboard, monitor, computer, printer, and microwave oven, built with standard components.

After she learns about the brand and type of Alice's printer (e.g. from Trudy the "janitor"), she can purchase the same type and analyse it for weak spots. She will try to measure residual emanations, electronic and acoustic, and find what parts of the frequency spectrum she needs to listen to. She will then measure the bandwidth around centre frequencies, and try to detect the type of residual modulation present (e.g. AM or FM). Now she can arrange a proper listening device, e.g. a directional antenna, a low-noise pre-amplifier, an RF receiver, and a proper demodulator. Since the signals received are usually very weak, she needs to adjust her filters to minimal possible bandwidths, to reduce noise as much as possible. The demodulated signal will probably have to be processed with various advanced digital filters (like e.g. a **deconvolution filter** used in 2.4.1), to get useful signals.

Let's start with a simple TEMPEST attack on a dot-matrix printer.

6.1.1 • TEMPEST on a dot-matrix printer

Dot-matrix printers are very bad from a security standpoint. Although daisy-wheel printers create a lot more acoustic noise, the noise created by hitting different typebars is only slightly different. This is why a TEMPEST attack on a dot-matrix printer is relatively easy - because the total number of pins hit simultaneously is roughly proportional to the amplitude of audible noise created.

Dot-matrix printers have several vertically arranged pins (usually between 7 and 24) in a printer head, on a horizontally travelling carriage. 24-pin types were the top of high-tech for home computers in the mid-1980s. The printer used in this demo is Samsung Bixolon SRP-275, with 9 pins in the head, printing on a standard 76mm wide, rolled paper tape. Eve will buy the same type on eBay and start with a few tests. First, let's scan the printed letters, to check the displacement of dots in a matrix (figure 6.1).

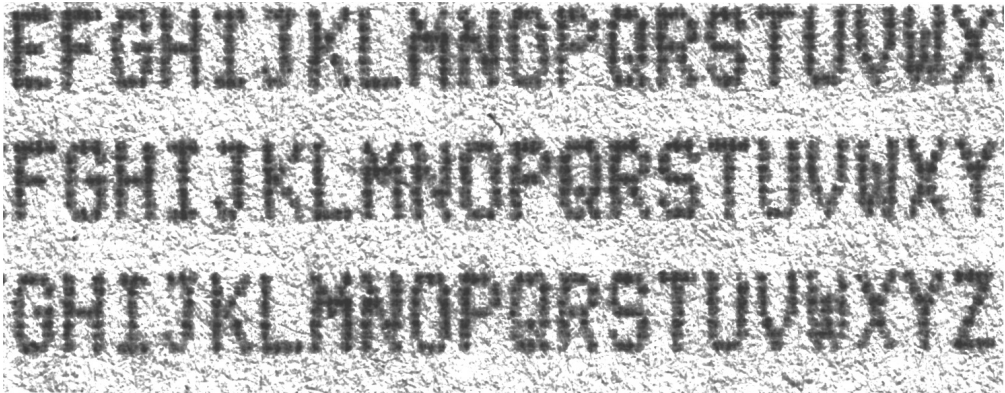


Figure 6.1 - SRP-275 printout, scanned, enlarged and enhanced

As you can see, all the uppercase standard ASCII letters fit in a 7x5 dots array. As the carriage travels left-to-right, printing e.g. letter “P”, first it hits 7 pins, then 2, then 2, then again 2 (now different pins). This way, measuring the amplitude of acoustic “bursts”, the letter “P” can be “encoded” as 7-2-2-2. Following the same logic, the letter “I” is encoded as 2-7-2, letter “L” as 7-1-1-1, letter “H” as 7-1-1-7, and letter “W” as 6-2-4-2-6. These are the letters with acoustic signatures that we will measure and compare.

Measuring the frequency spectrum of the device’s residual emanations (in this case acoustic) is the next step. Let’s start with the human-audible range. In figure 6.2. you can see a spectrum of a signal measured on the microphone input port of a PC, using a standard mic. There is a pronounced peak at 2650 Hz, so we start from there. This corresponds to the total number of horizontal dots per second that the SRP-275 can print.

There is nothing above 10kHz, but this is probably because of microphone and microphone input filtering. Nothing higher than 4kHz is needed for a human voice. Spectrum components deep inside the ultrasonic range (higher than 20kHz) can be expected because the printing pins are very short. Furthermore, analysing the 2650Hz bursts is difficult, because the acoustic ring-out of one vertical set of pins hit probably lasts too long, for the time needed to print at least 10 more horizontal dots. This is why the listening device will have to be tuned to higher ultrasonic harmonics (above 30 kHz) which ring out much faster. The adjacent horizontal sets of acoustic bursts need to be separated to detect the acoustic encoding pattern signature in the time domain(e.g 2-7-2 for letter “I”).

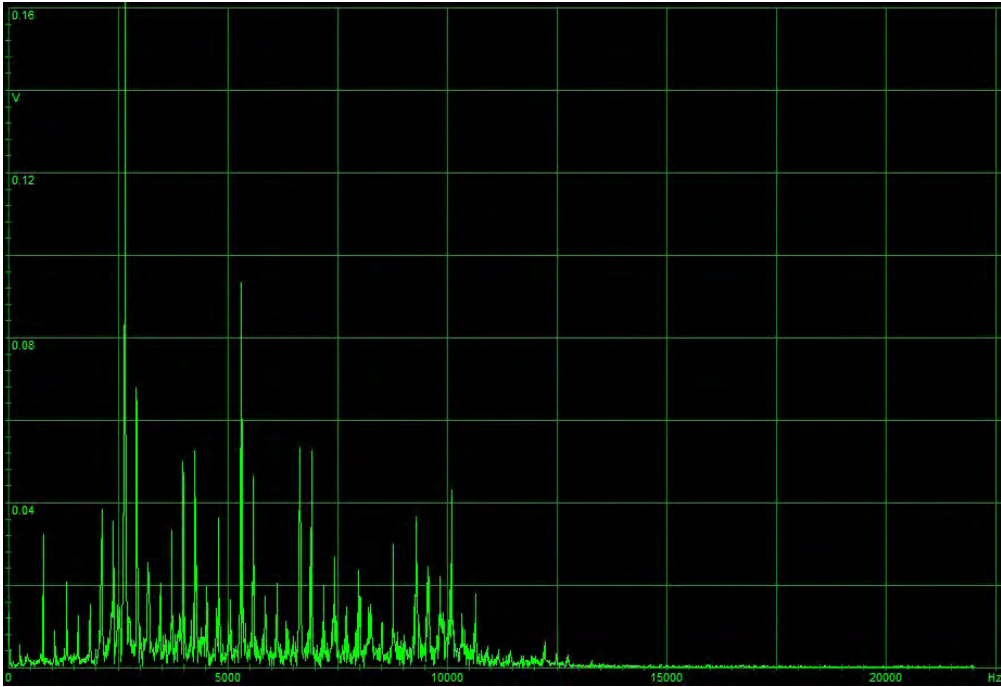


Figure 6.2 - SRP-275 human-audible spectrum

The TEMPEST receiver is shown in figure 6.3: A simple 2-stage amplifier, using electret mic MCE-2500 with an ultrasonic response up to 100kHz. Capacitors C2 and C4 define lower corner frequency at 30kHz, to filter out audible low-frequency components with too long ring-out.

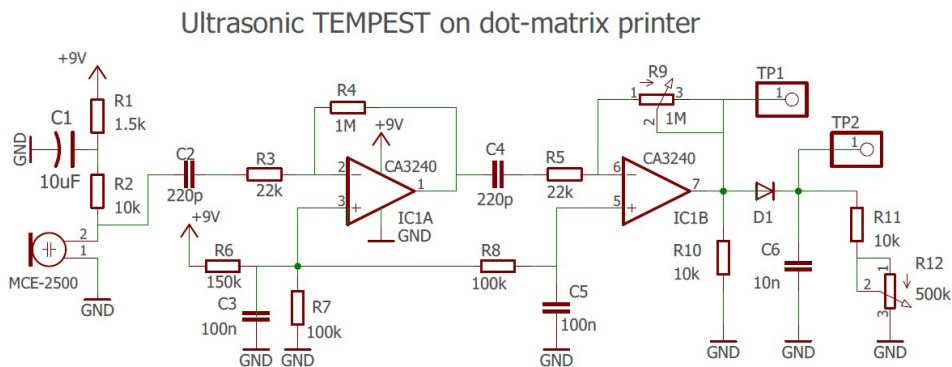


Figure 6.3 - Schematics of the ultrasonic microphone amplifier

The two test points are monitored on a digital oscilloscope. After some tweaking to potentiometers R9 and R12, audio signatures can be captured.

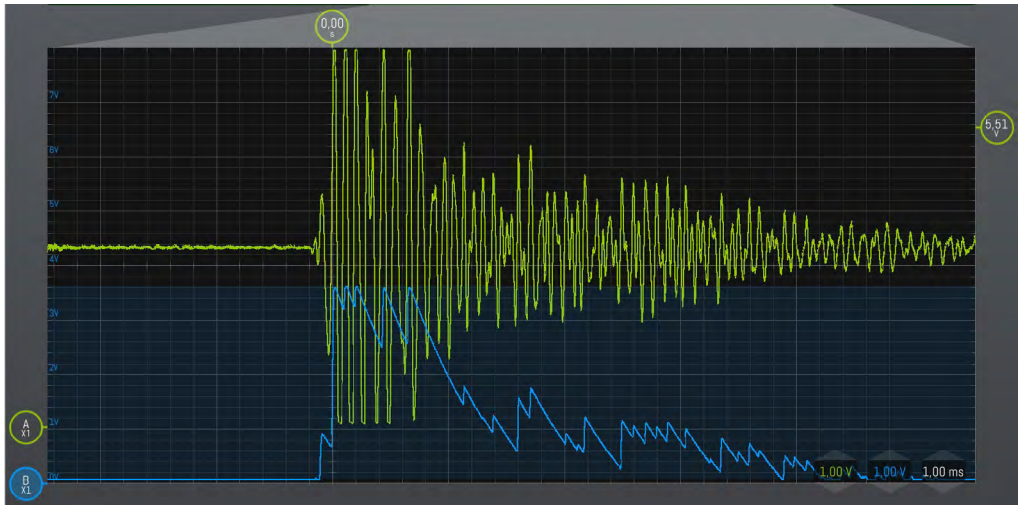


Figure 6.4 - Audio signature of the letter "L"

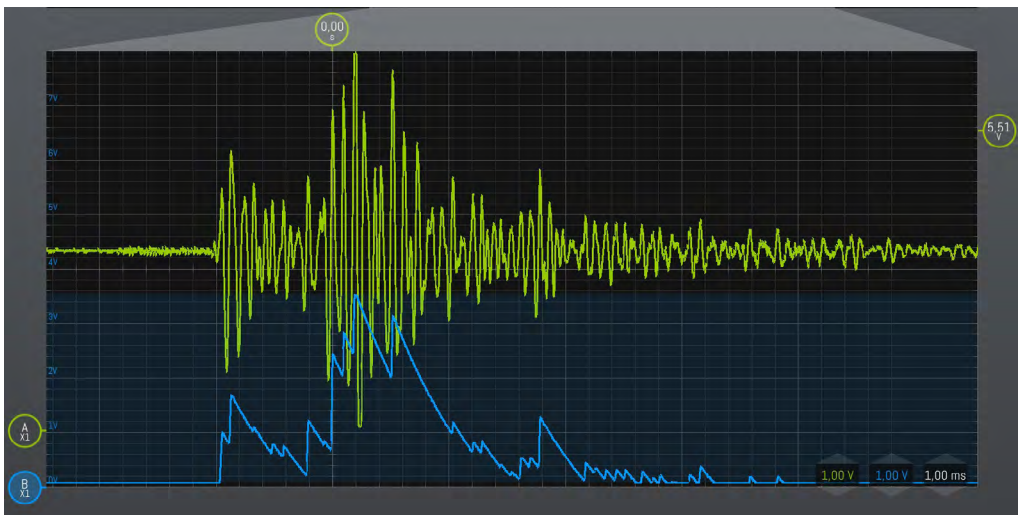


Figure 6.5 - Audio signature of letter "I"



Figure 6.6 - Audio signature of letter “H”. The amplitude was saturated-clipped at 7V, which also created too long ring-outs

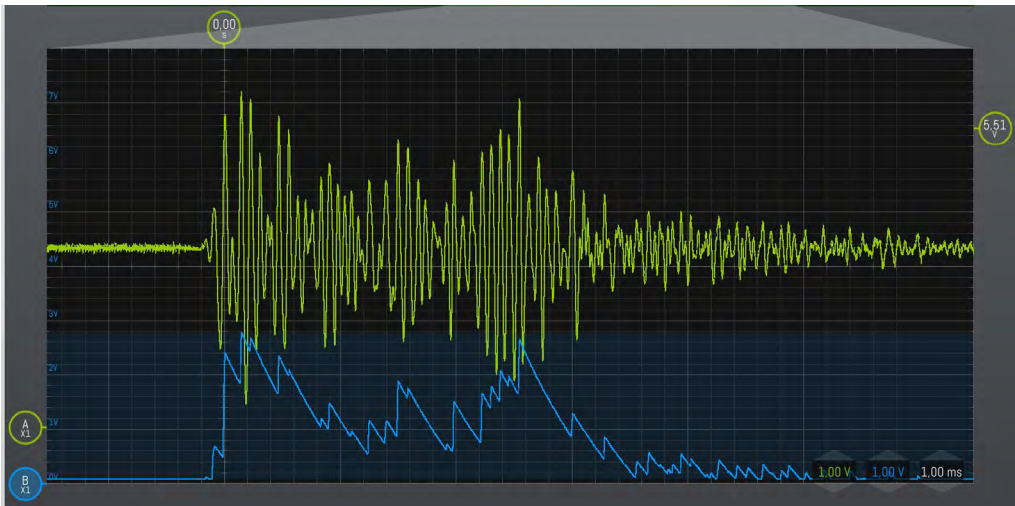


Figure 6.7 - Audio signature of letter “W”. Note two high peaks and one lower

Since the electronic hardware used was a very simple analogue amplifier, the results are far from perfect. The ring-out is still too long and strong, making it difficult to read some signatures. Anyway, this is good enough for a demo to show the basic principles. With much better band-pass analogue filters (like at least a 4th-order Butterworth filter), enhanced ultrasonic directional microphones, improved AGC (auto gain control - to avoid saturation clipping, like in figure 6.6), and more advanced digital deconvolution filters (like in 2.4.1, to cancel the ring-outs) it can all be much improved.

Much better digital processing can enable the recognition of all characters. Please note that some characters can have very similar, or even equal audio signatures (depends on the number of vertical pins hit simultaneously, but not on the actual vertical displacement of dots). If a printer is used to print plaintext, this is not a problem. The message will still be legible thanks to numerous redundancies in every known language, as analysed in section 2.2.

Dot-matrix printers may be obsolete nowadays, but they can still be used because of some of their properties: They are simple and cheap and don't need software drivers to print text. Just connect them to UART and send 9600-8-N-1 ASCII characters. This was just a simple example to start with. On the other hand, keyboards are used everywhere...

6.1.2 • TEMPEST on a PS/2 or an USB keyboard

At first, I was thinking about tuning a radio receiver to a PS/2 keyboard controller XTAL frequency, somewhere in the HF range (usually 8-30 MHz). Then I found that my keyboards tested (one PS/2 and one USB) also radiate in the VHF range, between 75MHz and 120MHz (slightly outside the standard VHF FM radio broadcast band). This frequency isn't related to keyboard scanning or PS/2 or USB communication port clock frequency.

So, let's start with a spectrum analyzer. I used the pocket-size "**TinySA**", a poor man's spectrum analyzer [36], with a 2.8" screen, battery, and USB port, that you can buy for less than 100 EUR. Cheap, but good enough for this experiment.

To detect frequency spans of RF emanations transmitted by a keyboard (or any other electronic device under attack), it should be disconnected from a computer and connected to a well-filtered 5V power supply. An old 50Hz transformer coil is preferred over a switch-mode type. This is because switch-mode power supplies generate interferences at much higher frequencies, which could interfere with spectrum analyzer measurements. When connected to a computer, the RF EMI generated by a computer may interfere with measurements. This must be done to indisputably determine that an RF radiation detected in a certain frequency span comes from the keyboard, not from a computer or some other source.

After connecting the keyboard to a filtered 5V power supply, and after some "hunting in the dark", I detected a weak signal at 82.6MHz, 10dB above the noise floor, with a span of cca. 100kHz. It disappeared when the power to the keyboard was turned off, so now I can be sure that the detected signal came from the keyboard.

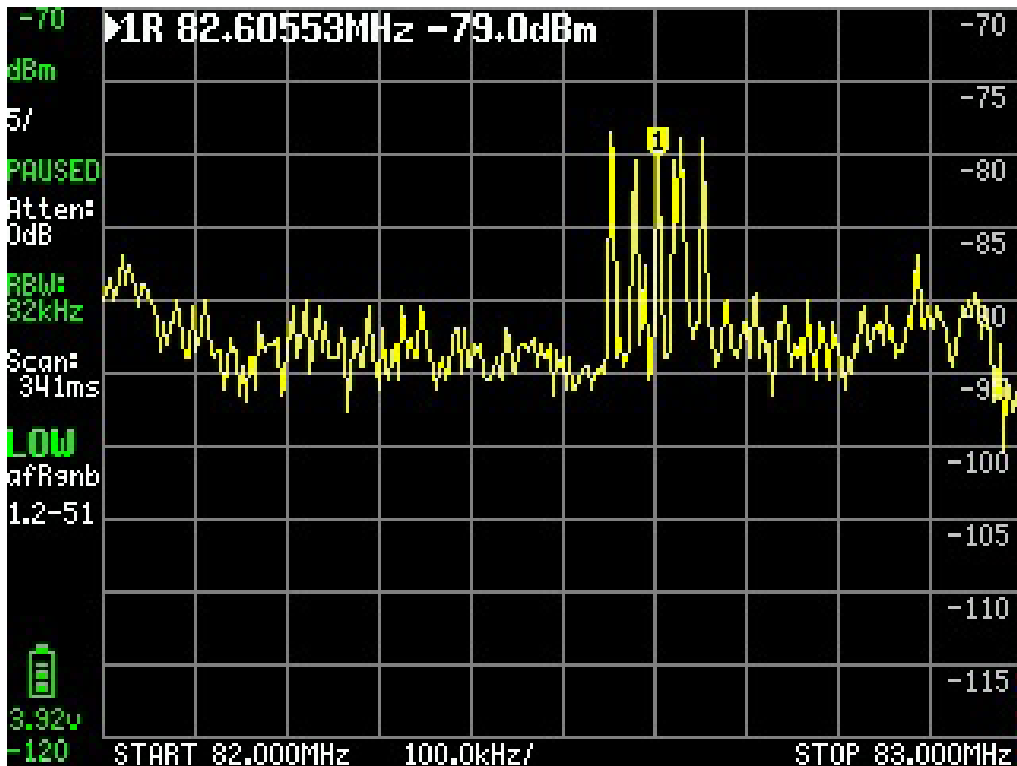


Figure 6.8 - TinySA spectrum analyser screenshot

When we know which RF frequency to listen to (82.6MHz), the keyboard can be connected to the computer (like in its normal working environment). I used a standard analogue FM radio receiver since it covers a range of 87-108MHz. You need to tweak the variable tuning capacitor to increase capacitance and lower the resonant frequency down to 82.6MHz. Of course, you can use an SDR¹ to adjust other parameters and extract a much better signal, but I used a standard analogue radio to prove that TEMPEST attacks can be mounted with very simple hardware.

After the radio was tuned to 82.6MHz, I connected its EAR audio output to the oscilloscope input channel (yellow trace). The other oscilloscope channel was connected to the PS/2 data line (blue trace). Eve would normally use an SDR, and try different baseband bandwidths and different demodulators (it is possible that pressing the keys performs amplitude, not frequency modulation), and then connect a pre-amplifier and a directional antenna to increase range. I didn't use any additional analogue filters or amplifiers here. Additional digital filtering is needed to improve the signals.

1 **Software-Defined Radio:** uses digital filters and digitally controlled circuits, parameters and also algorithms fully defined by software. Frequency bandwidths, carrier and baseband frequencies and methods of demodulation can be more widely and precisely controlled than on a standard radio, all through software, without a need to manually tweak on trimmers.

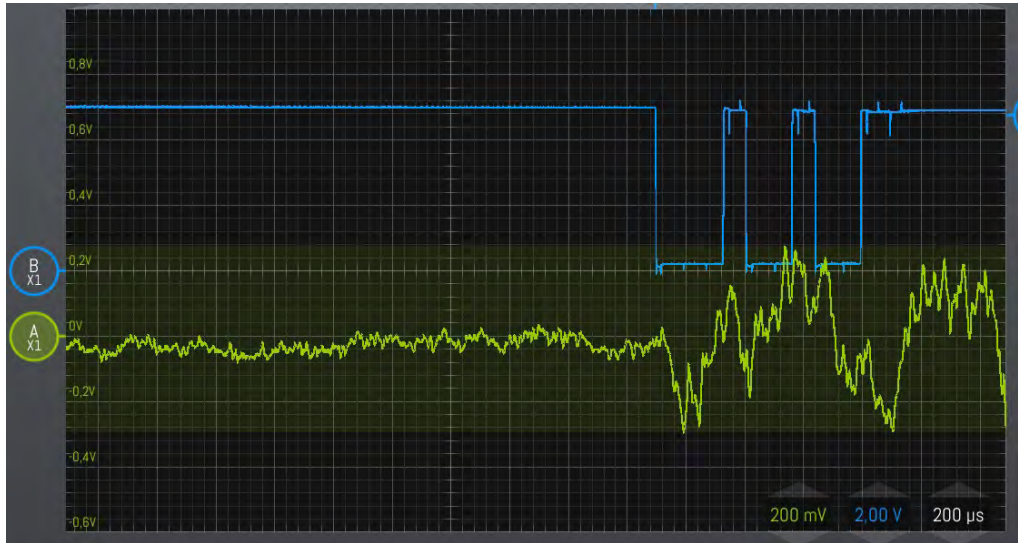


Figure 6.9 - PS/2 keyboard TEMPEST, key “E” pressed, scan code 0x24



Figure 6.10 - PS/2 keyboard TEMPEST, key “U” pressed, scan code 0x3C

The procedure for the USB keyboard is almost the same. The VHF RF signal is generated by a parasitic resonator similar to Wim van Eck’s attack on a VDU. The parasitic RF generated and modulated here is still much higher than the frequencies that the device under attack was designed to work with. Furthermore, changing the keyboard power supply voltage in a range from 4.5V to 5.5V shifts the RF frequency between 81MHz and 84MHz, which is another indication that it comes from a parasitic resonator. If it came from a stabilised oscillator it wouldn’t depend on power supply voltage. The RF generated by the USB

keyboard tested was around 105MHz which is difficult to detect with all the strong FM radio station signals around.

These two simple TEMPEST attack demonstrations were done to prove the concept. Hopefully, I have managed to spark your imagination, so you may try some TEMPEST attacks yourself. As you can see, you don't even need to know what the exact source of TEMPEST radiation is. **I still don't know exactly what created the VHF radiation** from my keyboards tested. If you get a usable signal after detecting the RF frequency span and processing the signal properly, this is good enough.

You can check any of your electronic devices for leakage. Then you can try to build better eavesdropping hardware, as explained before. You can also try sweeping through UHF, even if you detect HF or VHF carrier first. Maybe the device radiates also on a much higher frequency - it is easier and more practical to make a smaller directional receiving antenna for a shorter wavelength. The efficiency of parasitic "transmitters" (and consequently the range) is also much higher at UHF since the dimensions of radiating elements are usually below 1 metre. My receivers (RF-VHF for the keyboards and ultrasonic for the printer) were less than a metre away from the targets. With improved antennas, analogue and digital hardware, and algorithms, you will do well if you increase the range up to 20-50 meters. If it comes down to 2-5 meters behind a wall, from an adjacent room, it will be more than enough.

If you don't have an original idea, you can try [37] to get some inspiration. This is software that makes your monitor "radiate" audio-modulated shortwave RF, which can be received by any analogue shortwave receiver. Black and white stripes of different widths displayed on a monitor, changing with the music played, modulate different audio frequencies on the monitor's parasitic RF "transmitters". Check out [38] or [39] to get an inexpensive SDR start-up kit. Both kits come with books explaining SDR principles from starting with the basics, as good foundations if you want to delve deeper into the amazing world of TEMPEST.

6.2 • Buffer-overflow attack demos

They will be executed on a ZMC-Z80. For this to work, I will have to remove all **hardware protections** (figure 5.9) on the Crypto Dev Shield (described in 5.3.2 - watchdog timer-WDT, ROM-write alarm, and RAM-execution alarm), and software protections (UART input buffer length limitations) from UART input routine. The target is a simple system, but much less **vulnerable** to buffer-overflow attacks than any modern computer system (as analyzed in 2.3). All procedures executed here are the same (in principle) as when attacking a general-purpose PC or a web server. Their architecture is much closer to von Neumann's (like ZMC-Z80) than Harvard's. You can read [40] for more detailed information on buffer-overflow attacks.

This is an attack for Mallory, not Eve. The same as always, she will need to gather some information about Alice's system (ZMC-Z80 here). Millions of computers today run on the same operating systems, use the same software, and often even the same hardware, sharing the same weak spots (on all hardware and software levels) that can be attacked.

The basic information about many systems (including the ZMC) is thus almost always **publicly available**. Please also note, that today even **civilian and military systems** regularly use the same or very similar software and hardware (as explained in 1.1.4).

This way, by reading ZMC manuals, publicly available at [31], Mallory can learn the following basic information to prepare for an attack. **The memory map** is the most important: 64KB altogether, address range 0x0000-0x7FFF is ROM (actually EPROM), containing firmware code for driving the FP (Front Panel - scanning the keyboard, refreshing the LED display, buzzer, and FP UART port-bit banded by the code in ROM, single-stepper, machine code monitor and all the other low-level routines), a simple integer-number BASIC interpreter, FP UART-RAM program loader and many other useful routines. This code is stored in the address range 0x0000-0x364F. This leaves almost 60% of EPROM available to Alice to store her firmware there. Alice's firmware will be stored starting at ROM address 0x3700. We can assume that Mallory knows all about Lee Hart's firmware code (published on his web page), but not about Alice's code that she will add later. The address range 0x8000-0xFFFF is RAM. The bottom part 0xFE00-0xFFFF is used by Lee Hart's firmware to store important variables. The stack starts above 0xFE00 and "grows" upwards, towards **lower** memory addresses (stack pointer SP value on a Z80 system decreases as the stack grows). Address 0xFFED contains the last FP keyboard button pressed. Eight bytes in range 0xFF78-0xFF7F are important for the system to determine the type of reset that occurred. If they contain the following sequence: 0xF0, 0xE1, 0xD2, 0xC3, 0xB4, 0xA5, 0x96, 0x87, it is considered that the CPU was reset, but the SRAM was preserved (its Vcc power supply pin wasn't shut down) because it held the correct "signature". It is highly unlikely that these numbers would be there if the SRAM is powered up after being turned off (after the **power-cycle**², SRAM always contains some **quasi-random**³ values). If the correct sequence is not there, this is considered a COLD restart, and all SRAM and CPU register contents are considered invalid, so they are reinitialised.

The ZMC with FP board installed runs in IM1 interrupt mode. The /INT pin on Z80 receives an interrupt request from FP every 1.024ms, to refresh the LED display and scan the keyboard. If the interrupt is disabled (assembler instruction DI -opcode 0xF3), the LED display will turn off, and Alice will notice it immediately.

Besides the FP board, my Crypto Dev Shield board will also be connected (figure 5.9). The WDT, ROM-write, and RAM-exec alarm circuits will be removed, but other circuits will be operational. Its UART port (I call it CTC-UART because it is clocked by CTC on the shield) will be configured as a **user port**, with only **low-level privileges**, executing simple tasks

2 power-cycle: turning a device off and then on. Usually erases volatile RAM contents and fills it with some quasi-random values at power-up. If RAM IC is cooled down below -20°C, the power-cycle required to erase RAM may become longer than 30 seconds. This is a cold boot attack (described in 2.6.4), we'll do it later.

3 unlike pseudo-random numbers: which pass NIST testing (because their sequences have apparent random properties), quasi-random numbers only appear to be random at first glance. Their sequences (e.g. a sequence of values in SRAM after a power-up) would never pass NIST test. Initial values in SRAM depend on many non-random effects, such as small internal differences between individual bistable cells (a consequence of inevitable manufacturing process tolerances), but also the effects of long-term "burn-in" (described in 2.6.3).

for a remotely connected user. This is where Mallory can connect and try to attack the system. The FP-UART port is the **service port**, where Alice can connect locally and perform administrator's actions, which require **high-level privileges**. The FP-UART is connected directly to the local Alice's terminal, while CTC-UART is connected to the **LAN network** through a WIZ107SR, which is a UART-to-Ethernet converter. This way, the low-level user remote terminal is connected to LAN (may go through WAN also since the WIZ107SR supports TCP/IP) through another WIZ107SR.

Apart from removing the aforementioned hardware protections, I will have to remove software limitations to the CTC-UART input buffer from Alice's firmware code, to enable the buffer-overflow attacks. Originally, the CTC-UART routine which loads the input buffer with bytes being received on the CTC-UART port was programmed like this:

```
LD A, (0A002H)
LD C, A
LD B, 0
LD HL, 0A004H
ADD HL, BC
LD (HL), D
CP 255
JR Z, cont1
INC A
LD (0A002H), A
cont1:
```

The SRAM address 0xA002 contains a counter of bytes received and stored in the input buffer. The input buffer starts at address 0xA004. Register D contains the last byte received on CTC-UART. Register pair HL is first loaded with 0xA004 (start address of the input buffer). The number of received bytes is first loaded to the A register and then moved to the C register, while the B register is **always** loaded with **zero**. **This is very important**. It means that HL can't be increased by a value higher than 255 (0x00FF), after the ADD HL, BC command is executed signifying that the highest SRAM address that can be written to (with the following command LD (HL), D which writes the received byte to the input buffer, to the SRAM address pointed by HL register pair) is already limited to 0xA004 + 0x00FF=0xA103, thus limiting the buffer size to 255 bytes. The rest of the code (after CP 255 which compares register A to a constant number 255) keeps the value at 0xA002 from increasing (actually wrapping around to 0). Without it, the program would continue to write incoming values, starting again from address 0xA004. The buffer would still not overflow.

Simply removing the protection after "CP 255" is not enough to enable the buffer overflow. A complete rewrite of the CTC-UART input routine was required to make it susceptible to this kind of attack - now you see how Alice can create multiple levels of protection if she writes her code well.

In the following demonstrations, Alice will connect using "Tera Term" terminal (the nostalgic green-on-black). Mallory will use "Br@y's" terminal (the grey one) from a remote location.

Low-level users can connect to the CTC-UART (through Ethernet LAN or even WAN network if needed) after Alice gives a command “GO 3700”, which starts Alice’s CTC-UART port terminal firmware (appended to Lee Hart’s firmware at address 0x3700). The user port will signal it is ready to accept commands with “READY.”

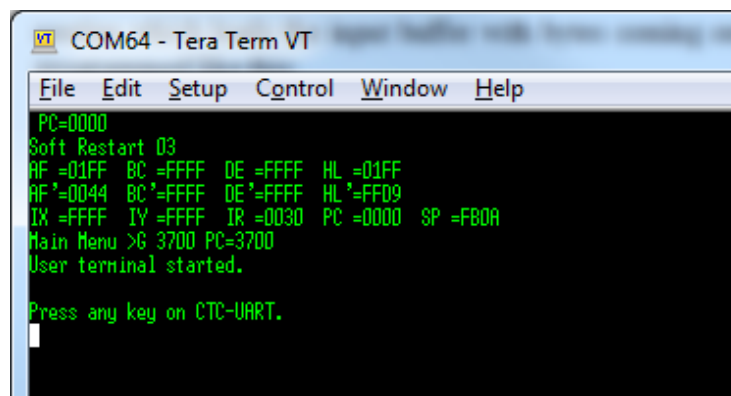


Figure 6.11 - FP-UART, admin terminal started

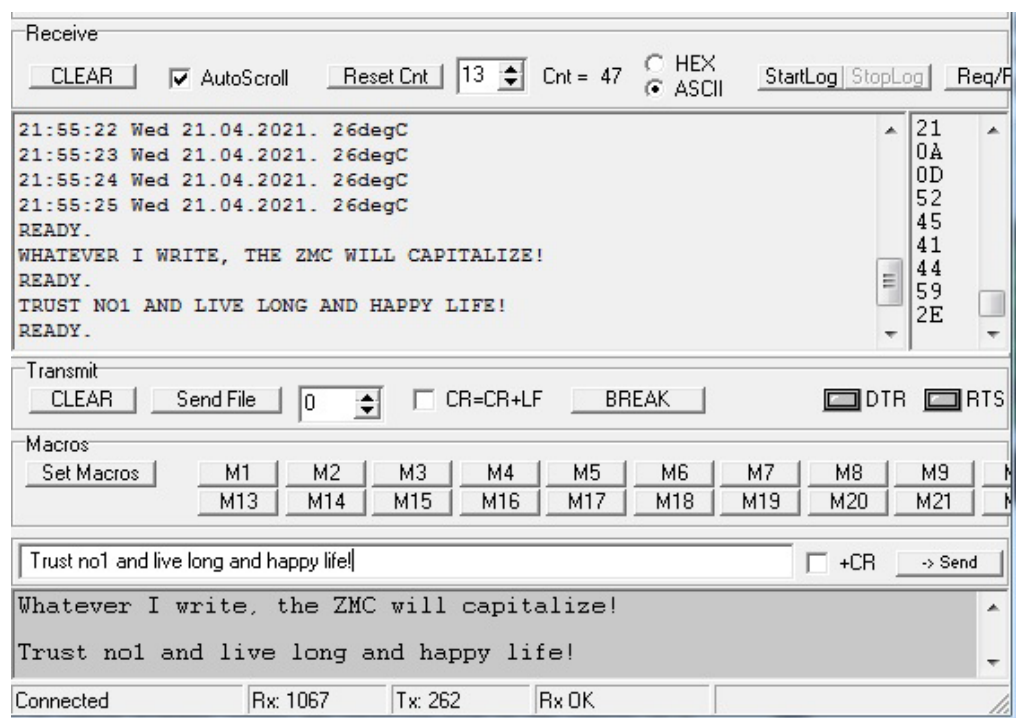


Figure 6.12 - CTC-UART, user terminal

Low-level users can perform only two actions, as programmed by Alice. They can send a sentence to the server, in which ZMC will convert all lowercase letters to uppercase and send it back to CTC-UART. If 0x23 (ASCII "#", hash) is entered, the ZMC will read time and date from the DS1307 RTC (on the Crypto Dev Shield) through the I²C interface, and temperature from the DS18B20 thermometer through the 1-wire interface and send it to the CTC-UART. Thanks to bad buffer-overflow protection (in fact none), Mallory can do a lot more.

6.2.1 • Smashing the stack on ZMC- Z80

Instead of a regular input (a sentence up to 255 bytes), Mallory will send a sequence of 32KB (32768 bytes) of zeroes (0x00). Mallory doesn't know the exact position of the input buffer (starting at 0xA004 address) but also she doesn't need to know it. Even if the buffer starts at 0x8000 (the beginning of SRAM), it will fill the SRAM with zeroes. If it reaches the end of SRAM, it will wrap back to 0x0000 (the beginning of ROM) where an attempt to write will have no effect. Take a look at figure 5.7 (the ZMC mainboard schematics). If a /ROM line is asserted (the most significant address bit A15 set to 0) along with the write request line /WE, neither RAM nor ROM memory chip will be asserted, hence no effect.

The bottom part of the SRAM will be filled with 0x00. This means the SRAM "signature" stored in 0xFF78-0xFF7F will be corrupt and will be interpreted as a "COLD" reset, and everything will start from zero. The stack (located above 0xFFE0) will also be filled with zeroes. This means when the CTC-UART input subroutine is finished, the RET command will pick a return address from the stack, which will surely be 0x0000. This will effectively execute a cold restart which is possibly enough to sabotage Alice's ongoing operation.

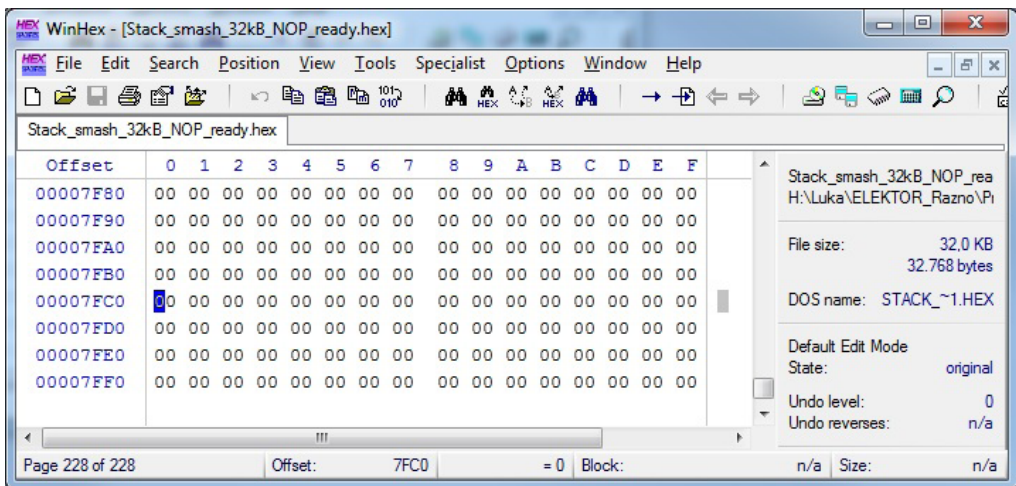


Figure 6.13 - Creating a 32KB file filled with 0x00

Instead of typing a regular input, Mallory will choose the "Send file" command on Br@y's terminal and send the previously prepared file.

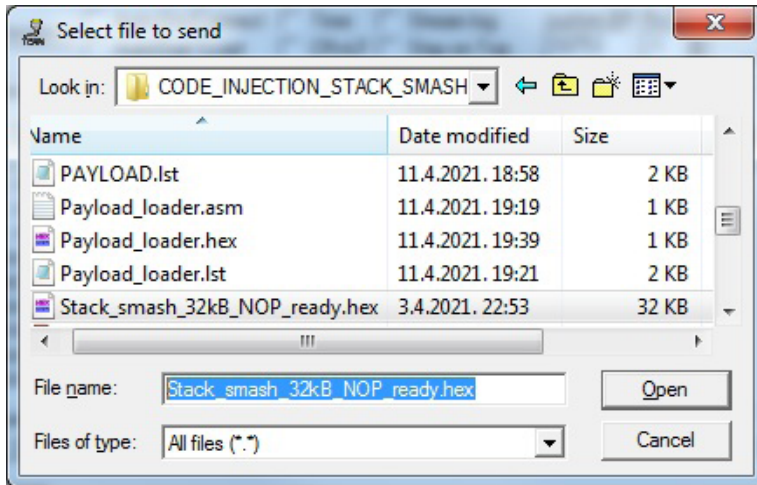


Figure 6.14 - Sending the 0x00 sequence

At the speed of 9600bps, the transfer of 32KB will take cca. 35 seconds. After the input is completed and the RET command makes a program jump to 0x0000 address, Alice will be stunned when her ZMC executes a cold restart!

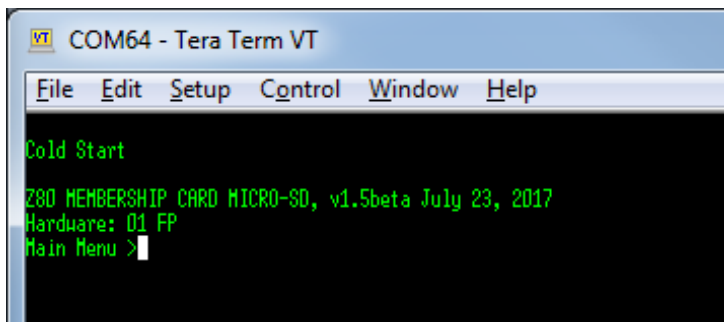


Figure 6.15 - The stack smashing attack was completed successfully!

Now you may logically ask if it is possible to insert a different address on the stack, other than 0x0000. The answer is yes, but the order of bytes may be a problem. The Z80 is an 8-bit processor, but has many 16-bit operations implemented directly. When storing a 16-bit number in SRAM memory (which has an 8-bit wide data bus), the Z80 does it in **low-high** order (also called **little-endian**, as opposed to **high-low** order, called **big-endian**). This means that LD (0x8010), BC command will store the register pair BC to SRAM, in a way that register C (the low-order byte) will be stored to SRAM address 0x8010 (lower address), and register B (the high-order byte) to address 0x8011 (higher address). The same happens when PUSH-ing a return address (a 16-bit number) on the stack or when CALL-ing a subroutine. Remember Mallory doesn't know either the position of the input buffer pointer or the SP (stack pointer). Let's suppose that she analysed Lee and Alice's firmware thoroughly, and concluded she could cause a lot more damage to Alice if she jumps directly to ROM address 0x6123 - the preparation procedure was explained in section 5.3.3.

Mallory will prepare a 32KB file filled with a 0x61, 0x23, 0x61, 0x23,... sequence. The problem is now that both input buffer pointer and SP can be set to **an even or odd value** (both are OK as SP, or any other memory address pointers on the Z80), and Mallory doesn't know any of them. This means, after sending that 32KB file, if she is lucky, (the chances are 50-50), the return address will be read as 0x6123, but it can also read as 0x2361 (low and high bytes swapped). This is a completely different part of ZMC memory! Since it is a common procedure to assign even values to pointers - because SRAM memory address range starts with an **even** number (0x8000) and ends with an odd number(0xFFFF), and the stack is always being filled **from bottom-up**, her chances are higher than 50% if she starts the sequence with low-order byte - 0x23, 0x61, 0x23, 0x61... On the other hand, if she finds a way to execute the attack by jumping to an address with two equal bytes (e.g. 0x3A3A, the same as with 0x0000), it will 100% succeed, but it is unlikely to find such an address because there are only 256 (less than 0.4%) such addresses in 0x0000-0xFFFF (64KB, or 65536-byte addresses altogether) range.

This order of bytes will be especially important in the next section, where Mallory will execute a more advanced attack, which will enable her to do a lot more, up to taking complete control over Alice's server! We will also see that Mallory's detailed knowledge of CPU machine code of the system under attack now becomes *conditio-sine-qua-non*, even more than for simple stack-smashing.

6.2.2 • Injecting and executing an arbitrary code

Instead of simply cold-restarting the system under attack, or a painstaking and long search through its existing firmware to find an address to jump to (Mallory has very little chance to find one), why not:

1. Inject an arbitrary code designed at your own will, and then
2. Execute it?

Sure, this can be done, but there are quite a few problems to be solved. This attack requires more thorough preparations. First of all, the code can be injected only to SRAM, (address range 0x8000-0xFFFF). The address to be written to the stack as a fake return address is chosen as 0x8080, for the reasons explained in the previous section. If Mallory prepares a *.hex file with 32KB of 0x80 bytes, it will surely fill the stack with 0x80 bytes and thus make a program jump to 0x8080 address, but there will probably be no useful code to execute there. This means the 0x80 hex file must be much longer than before, close to 64KB. This will load 0x80 bytes till the end of SRAM (address 0xFFFF) and then wrap around to 0x0000, "writing" to ROM with no effect until 0x7FFF, then continuing to write 0x80 bytes to RAM starting from 0x8000 address.

It would be ideal, to fill 0x80 bytes exactly till SRAM address 0x8080, and then continue with injecting Mallory's code (called the **payload**, meaning the useful attack code). This is not possible because Mallory can't know exactly where (to which exact SRAM address, the beginning of CTC-UART input buffer, which is 0xA004) she started writing to SRAM, so she can't know precisely when to stop. In this ideal case, the program would jump to 0x8080

(after a RET instruction picks it from the stack as a fake return address) and immediately start executing her payload attack code. This is why she has to make a much longer sequence of 0x80, to go **past** the 0x8080 address.

So what will happen now when the program jumps to the 0x8080 address? It will read 0x80, and pass it to the CPU instruction decoder, as a machine code instruction to execute. The Z80 manual states that 0x80 is the opcode for assembler instruction ADD A, B. This is a 1-byte instruction that takes the same time (4 CPU clock cycles) to execute as NOP. It adds the contents of register B to register A and stores the result in register A and sets various flags in the F register. This means that a long sequence of 0x80 bytes (a sequence of ADD A, B instructions) does practically the same as a sequence of 0x00 bytes (a sequence of NOPs, or a so-called “**NOP sled**”), spends the same amount of CPU time, affects only A, B and F registers, and most importantly, can’t make any program jumps. This is a reason why for example, a sequence of 0xC3 bytes can’t be used as a “**sled**”. Absolute program jump instruction JP has a 3-byte opcode (relative jump JR takes only 2 bytes), e.g. C312D5 means “Jump to address 0xD512” (little-endian, remember?). This means C3C3C3 would be interpreted as a command to jump to the address 0xC3C3, and the payload code wouldn’t be executed.

How long will the sled have to be? Mallory will have to make some “educated guess” here. If she makes it 64KB long, the injected 0x80 sequence will wrap around back to the beginning of the CTC-UART input buffer, and this may interfere with the operation of the CTC-UART input routine (some variables may be located before the beginning of the input buffer), which we want to be completed normally, so its RET command is executed - to pick a fake return address 0x8080 and jump there. This is why she will make it around 60KB long and give it a try.

4 sled: a long sequence of useless instructions (like NOP or ADD A,B here). The program execution will start at the fake return address planted to the stack (0x8080), and then “**slide down**” this sequence till it reaches some useful payload code to be executed.

```
.ORG 8000H                ; Payload starts at 0x8000
NOP
NOP
NOP                        ; 5xNOP in a sequence mark the beginning
NOP                        ; of the payload code to the loader
NOP
NOP
NOP
NOP
NOP                        ; Disable the /INT interrupt to keep Alice's
DI                          ; firmware from taking back the control
CALL 3DA4H                 ; Send the following text to CTC-UART
.DB 10,13, "Ready to serve my new master!",10,13,0    ; 0 is the end of a string
LD BC, 0                   ; load a constant for "wait" subroutine
CALL 3C94H                 ; Wait half a second
JP 8000H                   ; Jump back to 0x8000 and repeat!
.END
```

The payload code can then be injected after a sequence of 60KB of 0x80 bytes, but this brings another problem. Mallory still doesn't know at which exact address the payload code will start. This may cause some difficulties in writing the payload code (e.g. she won't know exact addresses for JP-jump commands). This is why she will first inject a short program: a payload loader, and then a **payload code** after the loader which will copy the **payload code** to SRAM, starting at address 0x8000. The payload code will be programmed and compiled to start at 0x8000 (using .ORG assembler directive). After the payload is copied, the instruction JP 0x8000 will be executed at the end of the **payload loader**, which will start the attack payload.

The payload loader will have to do some assembler programming trickery. The loader first needs to establish its position within the SRAM. The PC (Program Counter - the memory address of the next instruction to be executed) register in the Z80 CPU can't be directly read. This will be done in a way to CALL a short subroutine, and then read the return address from the stack. This will be the SRAM memory address of the next loader instruction after the CALL command!

As you can see, the loader program uses only two branching instructions: JR and DJNZ. Both are 2 bytes long, and use **relative addressing** to make a jump. They will work at any address. An absolute jump to 0x8000 is used at the end when the payload is ready.

The opcode for POP DE (take a 16-bit value from the top of the stack, store it to register pair DE, increase the stack pointer SP:=SP+2) from the Z80 manual is 0xD1. The opcode for PUSH DE (push a 16-bit value from the register pair DE to the top of the stack, decrease the stack pointer SP:=SP-2) is 0xD5. The opcode for RET-urn from a subroutine (to the address on the top of the stack) is 0xC9. These three opcodes are copied to addresses 0xFFFD-0xFFFF.

When this subroutine is CALL-ed, the address of the instruction following CALL 0xFFFFD (instruction LD H, D) is stored on the stack. POP DE will copy it to DE, then PUSH DE will restore the stack pointer, so then RET will return to the correct address. The address of LD H, D instruction will thus stay in DE. The loader will continue to search from this address (actually through its own program code!) until it finds a sequence of 5xNOP. The HL pointer then needs to be adjusted, for it to point exactly at the beginning of the payload code. The LDIR block move instruction is then initialised and executed. JP 0x8000 then starts the payload code!

```
.ORG 0000H      ; This is actually not important, the loader must work at any address!
    DI          ; Disable interrupts
    LD SP, 0FFA0H ; Initialise the stack pointer to 0xFFA0
    LD A, 0D1H
    LD (0FFFDH), A ; Load "POP DE" opcode 0xD1 to SRAM address 0xFFFFD
    LD A, 0D5H
    LD (0FFFEH), A ; Load "PUSH DE" opcode 0xD5 to SRAM address 0xFFFFE
    LD A, 0C9H
    LD (0FFFFH), A ; Load "RET" opcode 0xC9 to SRAM address 0xFFFFF
    CALL 0FFFDH   ; Call the subroutine - POP DE, PUSH DE, RET at 0xFFFFD
    LD H, D       ; DE now contains the SRAM memory address of this instruction!
    LD L, E       ; Copy DE to HL
    LD B, 5       ; Initialise the NOP counter

loop00:          ; Find a sequence of at least 5xNOP - this is the beginning of
    INC HL       ; the payload code injected to SRAM after this loader
    LD A, (HL)
    CP 0         ; Is the instruction read at (HL) address actually NOP?
    JR Z, cont00 ; If yes, don't reset the B counter to 5
    LD B, 5

cont00:
    DJNZ loop00  ; If B came down to 0, the 5xNOP sequence is found
    DEC HL
    DEC HL       ; Now it's time to copy the payload code to 0x8000
    DEC HL       ; HL now points exactly to the beginning of the payload code

    LD DE, 8000H
    LD BC, 200H  ; Increase if the payload is longer than 512 bytes
    LDIR        ; block move: (HL) ->(DE), INC DE, INC HL, DEC BC, until BC=0
    JP 8000H     ; Time to start the payload !!
.END
```

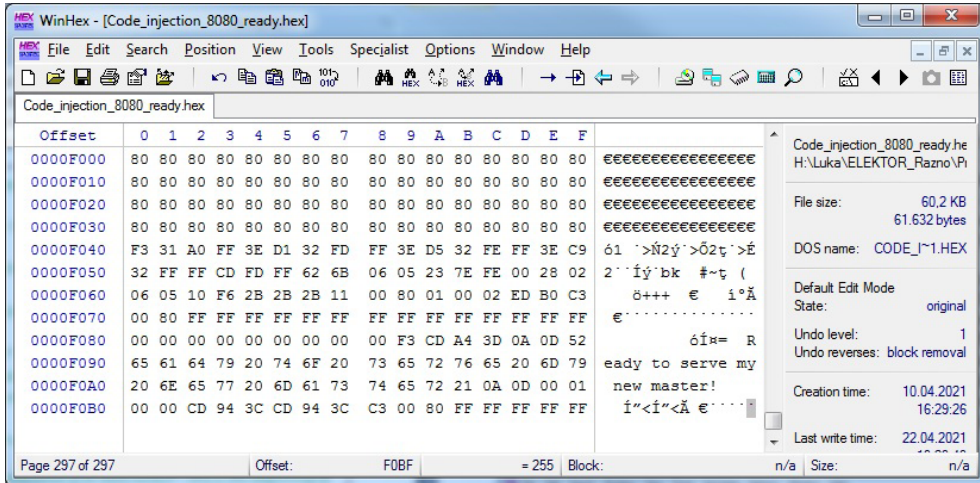


Figure 6.16: 0x80 sled, payload loader, and payload in a hex file, ready for attack

As you can see in figure 6.14, the file to be injected to CTC-UART is prepared in the following way: First, a *.hex file is filled with 60KB of 0x80 bytes. Then the payload loader is appended to this, from address 0xF040 until 0xF07F. The payload itself comes after the loader, from 0xF080 until 0xF0BF. The code injection attack file is thus exactly 61632 bytes long.

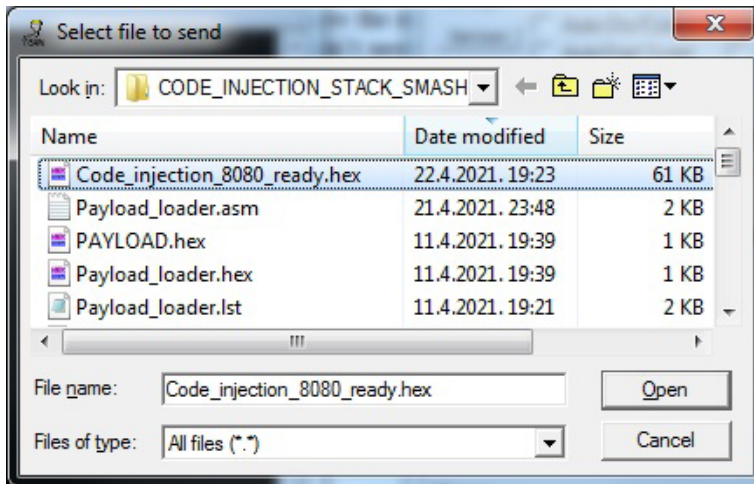


Figure 6.17 - Sending the code-injection attack file

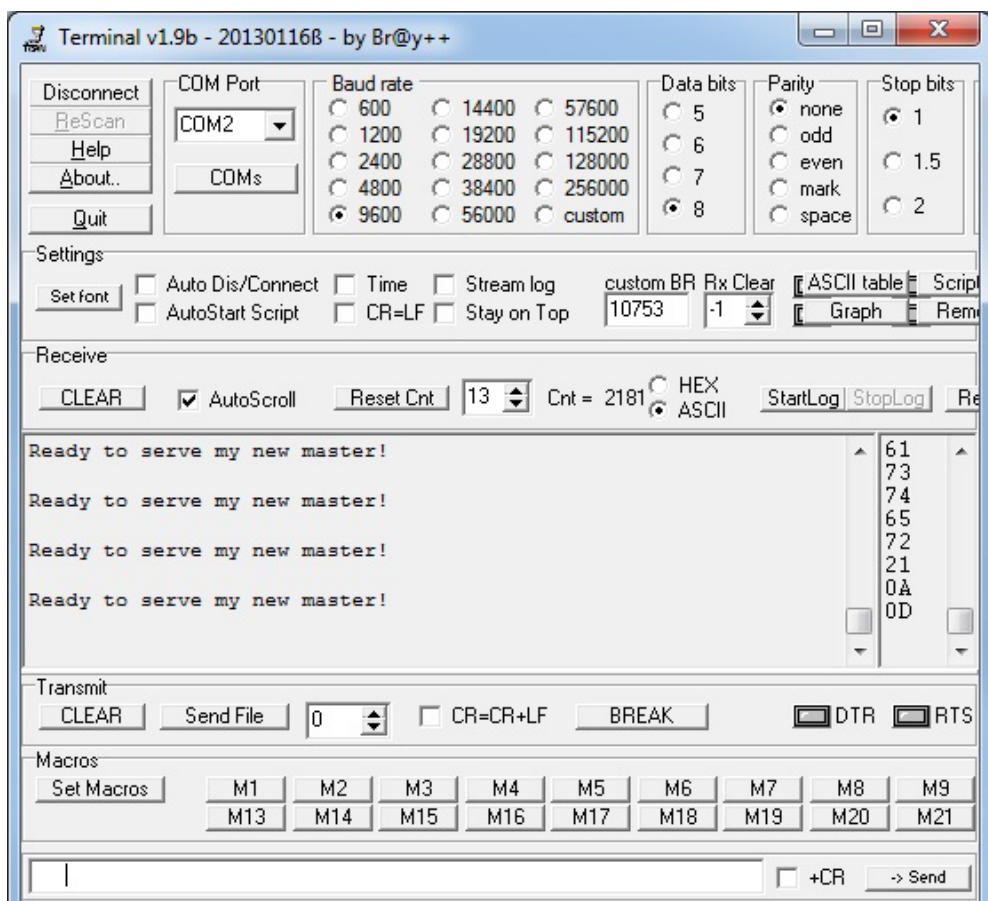


Figure 6.18 - The code injection attack was completed successfully!

Procedures to attack more complex systems are similar to this, and they are all more vulnerable by design than the ZMC Z80. I strongly recommend reading [40] for more detailed information. If you think this is too “hardcore”, please read “The story of Mel, the real programmer” [20], and also the entry “**Real Programmer**”.

The term refers to a programmer (by the standards of more than 20 years ago) who never uses a HLL (not even C), sometimes not even assembler mnemonics (like “JR loop1”), but hex (0x18F0 or even binary!) opcodes directly (ouch!). He can remember every hex and binary opcode of every CPU he has ever programmed. Apart from tricks I already presented in this book (like jumping in the middle of a 3-byte instruction opcode to execute a different instruction in 5.3.3 or using parts of the look-up table as opcodes in 4.2.1.2) that require good knowledge not only of assembler mnemonics but their hex opcodes as well, they sometimes pull “crazy” tricks like changing a single bit in an opcode (stored in RAM), to change not only an operand **but instruction as well**. This was described as a possible failure mode on an old EPROM caused by hot carrier damage in 2.6.2 (The Z80 example - opcode 0x77 - “LD (HL), A” changed to 0x76 - “HALT”, and possible runaway

code chaos as a consequence). If the same procedure is used to execute something meaningful **intentionally**, needless to say, it will be next to impossible to understand for a different unfortunate programmer (even a “real” one!) who tries to do some work on “real programmer’s code” later, since above all this, they never write manuals, of course. This is something Mel used to do when writing his programs mentioned in the story.

The Dictionary (edition is more than 20 years old now) says “Very few **real programmers** are still in existence **today**”. All I can tell you, even though assembler programming is definitely out of fashion, is that both Mallory (to execute successful buffer-overflow attacks), and Alice (to effectively defend against them) **must** inevitably grasp some “real programmer” knowledge and attitude if they both want to do their respective jobs! A real programmer’s work 30 years ago was about writing highly efficient code, optimised for speed (to run as fast as possible on a Z80 at 4MHz) and memory (to use the least possible of usually less than 64KB ROM and RAM available back then). **As you have just seen, the same approach can be used today, but with a different goal: to improve security.**

The last practical application of “real programmer’s” knowledge that I know about was at the turn of the century, while MCUs with integrated one-time programmable EPROM (EPROM without UV lens, like the one used for MyNOR CPU-less computer at 5.6, but integrated inside the MCU - no possibility of erasing binary zeroes back to ones, but still possible to electrically program ones to zeroes) were still around - EEPROM variants were much more expensive back then. If you wanted to update an existing program inside the MCU, it was possible in the following way: If there was enough unused memory (containing all 0xFF bytes), you could store the program patch there. Inserting a jump command (to the existing code) to jump that patch inside the existing program was a “real programmer’s” true challenge! He had to write down the hex opcode of the jump instruction first, then find an instruction in the existing opcode (in an appropriate position inside the existing program, of course) that could be **converted to a jump instruction to an appropriate absolute memory address** (i.e. the beginning of the patch code) **by changing some of its binary ones to zeroes** (reverse is not possible without a lens and UV light). After appending the patch to the existing firmware (like we patched up the code-injection attack firmware of 3 parts), using some raw hex editor, and changing that existing instruction to jump, the complete hex file could be burned to EPROM using a regular procedure, thus updating the firmware.

6.3 • SRAM burnt-in data recovery

The theory behind it was discussed in 2.6.3. It is time for a practical demonstration. The test rig schematics are shown in figure 6.19. The two-stage analogue amplifier (first stage Q1 common-base, then Q2 common-collector, a setup for maximum possible bandwidth for fast signals) amplifies voltage drop on R1, for measurement of SRAM supply current by oscilloscope on test point TP1. The whole rig is controlled by one ATmega8 MCU (not shown on the schematics). The address bus is set to a memory address which is to be tested. This is relatively slow, using two 74HC590 8-bit counters. This is OK, since the address doesn’t need to change quickly. The Data bus is read/written by the MCU, which also controls the /RD and /WR commands to the SRAM. These two control lines are used as triggers for

oscilloscope measurements, at test point TP3. The resistor array R2-R9 decouples the data bus, in case it is asserted simultaneously by the MCU and SRAM.

The resistor array R22-R29 is used to pull the address bus up or down (chosen by SW1), for measurements of read access and data bus rise/fall times through test point TP2. Octal buffer IC2 is used to drive the array of 8 LEDs to enable easier monitoring of the data bus. The key variables to be measured are the power supply current during read/write from/to SRAM, and voltages (actually delay/rise/fall times) on the data bus.

Changes in signals measured will indicate the burn-in effects, and hopefully enable the extraction of the bytes written before the SRAM was powered down. The main condition is that the SRAM cells have been holding constant values for a relatively long period. A good starting point to get several approximate time periods is table 2.2. of section 2.6.3. These times can vary, depending on the different SRAM ICs used.

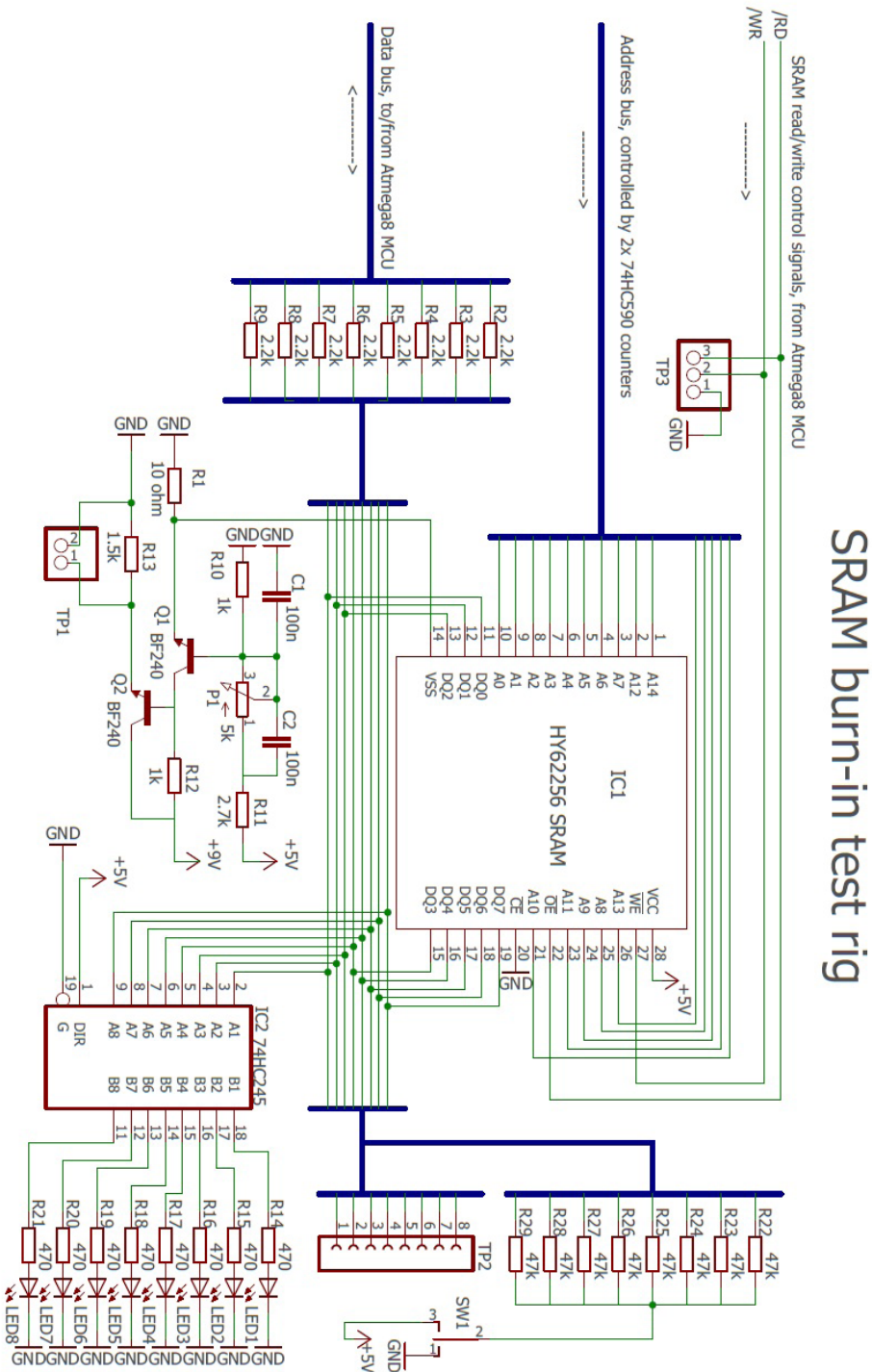


Figure 6.19 - Schematics of an SRAM test rig. The DUT is HY62256- a 32KB SRAM

One 100 Ω /5W resistor is attached to SRAM IC for heating, with a temperature sensor in between, so the SRAM can be tested at elevated temperatures, up to 80-90°C. You can see it on the test rig photograph, in figure 6.23. Changes in rising/falling times of 1-2ns need to be reliably measured, so a 100MHz oscilloscope is required. I used a **Tektronix-466**, equipped with an analogue screen-storage option (variable persistence screen with “flood guns”, there was one nice article in Elektor about this type of oscilloscopes in the Retronics section), for fast non-repetitive signals. This is not necessarily required, since the read/write test sequences can be made to run in repetitive loops, controlled by the MCU. I bought my Tek-466 in Ljubljana (Slovenia) at an electronic flea market, for 150EUR.

The following are some basic oscilloscope measurements:

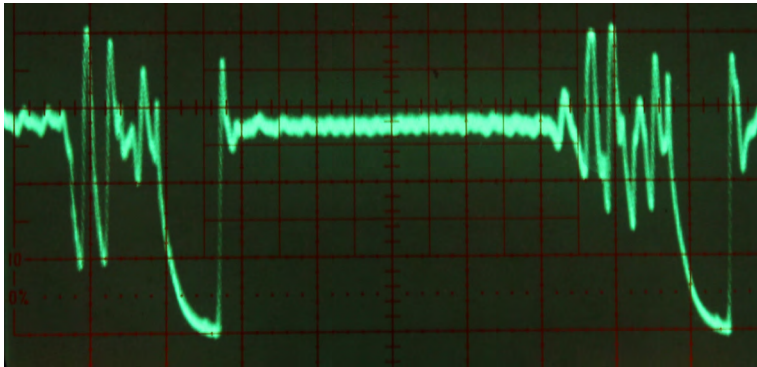


Figure 6.20 - Idd current, writing bytes 0xDE (left) and 0x01 (right) to SRAM (at 200ns/div)

Measurements of power supply current when writing from and/or reading to SRAM to cells with burnt-in data (those which have held the same byte for a long time) or without it (those who have had all of the bits in both states for an equally long time), and comparing the results can be used to recover the data. Comparison of read access times and waveforms (figure 6.22) between bits on the data bus can also help to recover data.

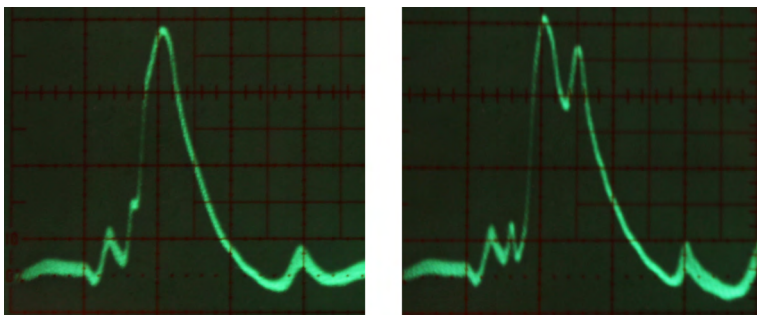


Figure 6.21: Idd current, reading 0xDE (left) and 0x01 (right) from SRAM (at 200ns/div)

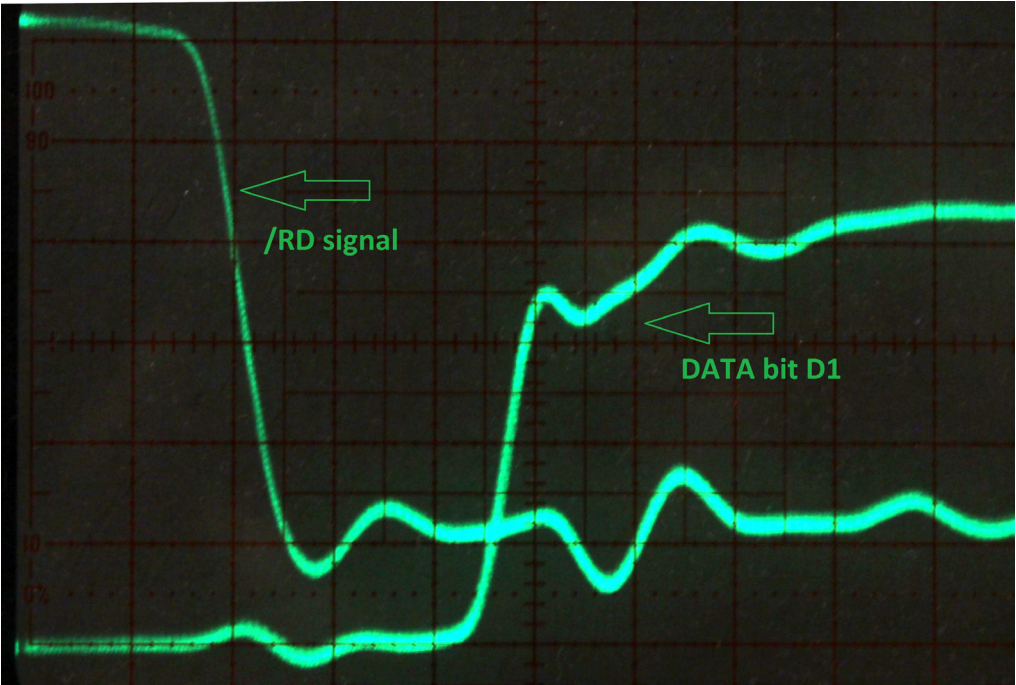


Figure 6.22 - Waveforms of /RD line and one data bit when reading from SRAM (at 10ns/div)

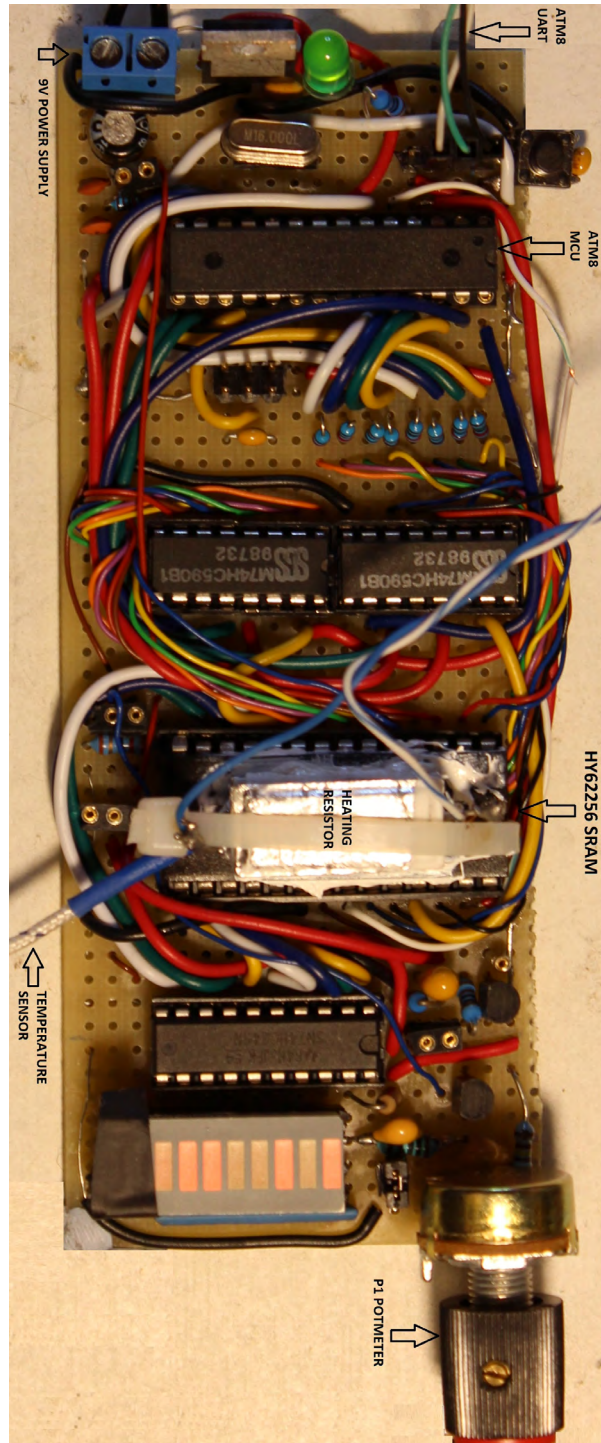


Figure 6.23 - A photograph of SRAM burn-in test rig

I wrote a constant string to SRAM, and left it to leave all the burn-in effects. I programmed other SRAM cells as counters, incremented every 10ms. Then I heated the SRAM to 80°C (to increase the burn-in effects by increasing the percentage of hot carriers, based on table 2.2) and left it powered up for 12 hours, while the MCU kept incrementing counters, to avoid burn-in effects on those cells. 12 hours later, I switched the heater resistor off and let the SRAM cool down to room temperature. The following are the results of the tests, on two SRAM cells:

- at address 0x204F there was a constant value 0x66, that we expect to have left some burn-in trace.
- 0x7FF1 was a constantly incrementing counter, without any burn-in trace.

Measurements of read-access times (like in figure 6.22) showed very small differences between 0 and 1 bits, less than 1ns, so I didn't consider them relevant. Only the MOSFETs used for bistables (that hold a 0 or 1 bit) are affected by burn-in effects, not the other MOSFETs (see figure 2.11) used to access the SRAM cell and transfer it to the data bus when reading. However, measurements of I_{dd} -power supply current while writing different bytes to burnt-in cells (figure 6.20), gave better results. This method is better for other setups (e.g. for reading SRAM from MCU) because it doesn't require physical access to the 8 bits of SRAM data bus.

Once again, the cell with address 0x204F contained the byte 0x66 and experienced a burn-in at 80°C. The other cell, with address 0x7FF1, had no burn-in (all bits being regularly and timely flipped). The measurements show that writing a byte pattern with more zeroes to a burnt-in memory cell takes more power than writing the same pattern to a cell without a burn-in. The difference in power required when writing 0x66 (a burnt-in value) and 0x99 (one's complement of a burnt-in value) to a burnt-in cell is much higher, compared to the other cell with no burn-in. These measurements alone are not enough to establish the exact byte pattern that was burnt into the cell. They require mathematical post-processing (filtering, correlating to known patterns, etc), similar to the TEMPEST attacks described in section 6.1. The 62256 is an old, reliable device without much propensity to burn-in effects, nowhere near new highly-integrated devices.

Many more experiments with other types of SRAM, under different operating conditions, are required. According to [13], some authors reported contradictory results, which subsequently indicates that residual burn-in effects aren't well researched and understood. This opens another new and wide field for further research!

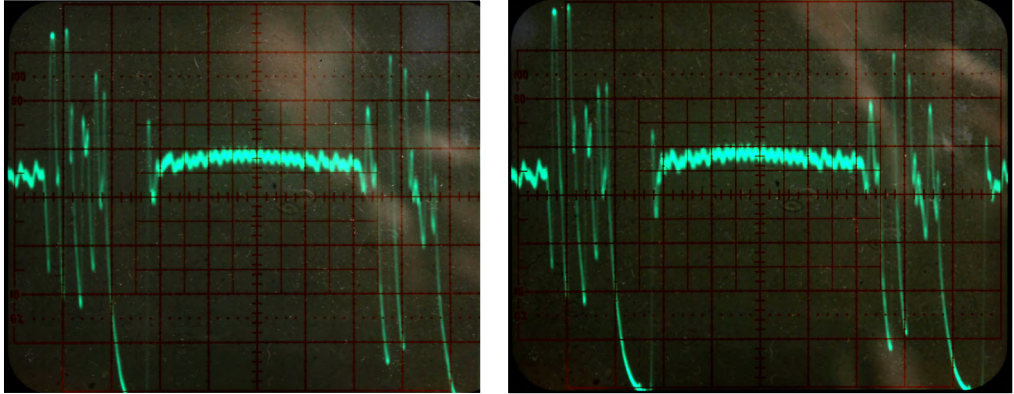


Figure 6.24 - Idd, writing bytes 0x00, then 0xFF to 0x7FF1 (left) and 0x204F (right)

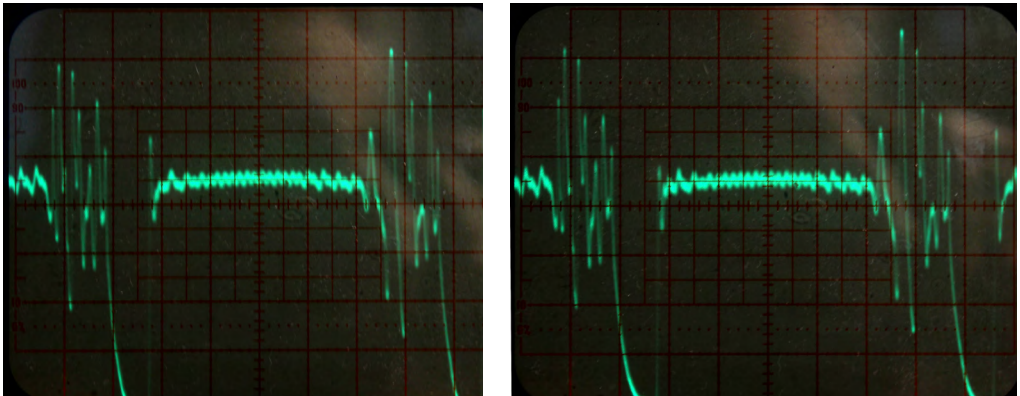


Figure 6.25 - Idd, writing bytes 0x66, then 0x99 to 0x7FF1 (left) and 0x204F (right)

Besides these aforementioned methods, there is one more good method of extracting the burnt-in data that I haven't tried yet. This is **gradually lowering supply voltage** down to the point where you get a wrong reading from a certain memory cell, or when writing a byte to it, it fails. Burnt-in cells and cells without burn-in are constantly compared. Usually, only certain bits in a burnt-in memory cell fail to be read/written correctly (e.g. when writing 0x00 or 0xFF), depending on their burnt-in state (0 or 1). This happens because **MOSFET gate threshold voltages** (V_T - see section 2.6.3) of burnt-in cells are always slightly increased/decreased, compared to cells without burn-in.

As far as Alice and Bob are concerned, they can work this into a secret communication channel enhanced with advanced steganography using other newer types of highly-integrated SRAM, more susceptible to burn-in effects than the 62256 tested here. An SRAM chip chosen for this setup will be OK if it can achieve significant burn-in retention after a few hours of heating at 80°C (while powered on, with secret encrypted message stored) if it can retain the burn-in effects for some 10-15 days when powered off (actually removed

from the circuit) at room temperature. The communication method would work like this:

1. Alice encrypts the secret message for Bob, then stores it on SRAM.
2. Alice heats the SRAM (while powered on, holding the secret encrypted message) to 80°C for 6-12 hours.
3. Alice lets the SRAM cool down to room temperature, while powered on.
4. Alice removes the SRAM from the circuit when it has cooled down.
5. Alice puts the SRAM chip in an envelope and mails it to Bob.
6. If Eve intercepts the mail, it just looks like some electronic components package from RS. Even if she tries to recover the burnt-in data, it still won't look very suspicious, because the message is encrypted. It doesn't mean Alice burned that data in on purpose. Maybe the chip was just removed from a server where it had been holding some constant data for a very long period - like a few months at 30°C.
7. Bob receives the mail (within a few days) and performs the data recovery procedure like already described in this section.
8. Bob decrypts the message.

Can you think of any other ways of how Alice, Bob, Eve, Mallory, Trudy, and Walter could exploit the memory retention effects? There are many!

6.4 • Cold-boot attack demo

This is the last demo, which is much simpler than the previous. This type of attack, as reported in [15] is likely to work against many old and new types of RAM. As explained in 2.6.4, cooling a chip down to even -50°C will not significantly reduce the conductivity of moderately-doped silicon micro-circuits, so the chip will continue to operate normally, but will retain the data after disconnecting the power because the discharge of MOSFET gate capacitances will be much slower. The same rig will be used for the experiment, only without the heating resistor. The 8 LEDs on the bar graph will display various running-light patterns directly from SRAM. If they continue to run in regular patterns (after a power cycle) it means the SRAM contents were preserved. If irregular, erratic patterns appear, it means the SRAM contents were lost. I will measure the maximum retention time at cca. -30°C, which is easy to reach using "KÄLTE 75" spray.



Figure 6.26 - Sub-zero cooling spray used in the cold-boot attack

At room temperature, there is only 0.1s retention time, but 10s can be expected at -30°C . The SRAM IC is first cooled down to cca. -30°C and the power-cycle time is gradually increased while maintaining the temperature with the spray. The 10s retention time was reached at -40°C . These results were to be expected since the power supply pins remain “shorted” through some $\text{k}\Omega$ -range resistance while powered down, so the Vdd pin may be considered as “grounded”, according to table 2.3.

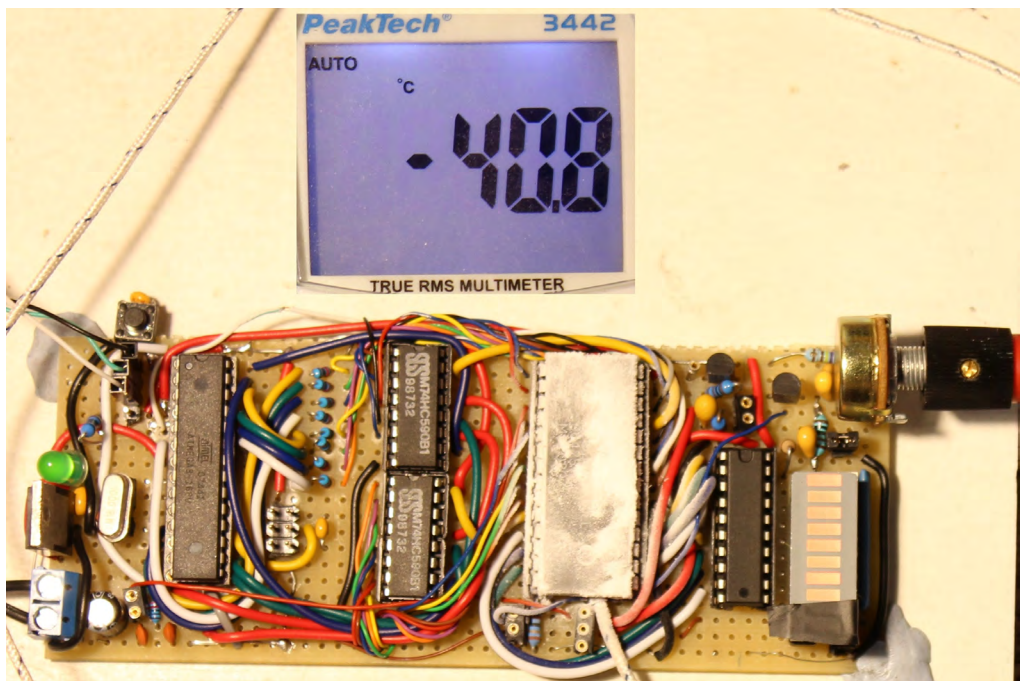


Figure 6.27 - SRAM was deeply frozen, to achieve the 10s retention time

As you can see, some practical attacks can be fully accomplished with very cheap equipment inside your home lab (like cold-boot and buffer-overflow attacks), while some other attacks (SRAM burnt-in data recovery and TEMPEST attacks) may need more sophisticated hardware and mathematical post-processing (although within the low-budget spies reach!) to be fully successful in real life, not just as proof-of-concept demos.

Please keep in mind that new highly-integrated electronic systems are **more susceptible to any of these attacks** demonstrated (except maybe for modern laser and ink-jet printers compared to old dot-matrix or daisy-wheel, considering the **acoustic** TEMPEST attacks. Their RF **electronic** TEMPEST signatures are another story, of course). Improving the ideas shown here may lead you to design your unique attack devices and procedures. There is plenty of work to do and no more excuses for 21st-century design engineers' typical apathy and depression!

I am sure that it can be completed on a low-budget nowadays. As the technology advances and becomes cheaper and more accessible, the process works much in favour of Alice and Bob. The next chapter will present some new ideas to work out, none of which I have tested in practice. They are still in a Matlab or LTSpice simulation stage, or yet to be worked out.

During the Cold War, the BND and CIA managed to infiltrate the supply chain of an East German military electronics manufacturer. They prepared rigged-hardware trojanized versions of ICs produced in the Soviet Union and East Germany and planted them on their suppliers. The IC housing design, Russian Cyrillic designations, housing shape and colour looked exactly as the original.

Despite all efforts, the rigged ICs immediately looked suspicious to a factory technician who was assembling a crypto-device. He sent them to their lab for inspection, where decapsulation was performed, and hardware Trojans were detected.
What went wrong?

Chapter 7 • A few more ideas to work on

There are many more ideas and cheap, simple ways to help Alice and Bob that I haven't even begun to develop. This final chapter is all about them. Ideas (the same as developing fully working prototypes) don't bring in much cash these days, so I will share them with you here. If you think you can find a way to commercialise some of them, please do your best!

7.1 • SIGSALY-2 “Reloaded”

The original SIGSALY dates back to WWII. It was a voice encryption system, based on one-time pads. It used a complex combination of 1940s analogue and digital (!) electronics, with **turntable records** (now is a good time to check [29] again!) used to store OTPs as recorded white noise. The whole system was extremely complex, weighed some 50 tonnes, and cost one million USD in 1943. (cca. 15 million USD today, corrected for inflation).

While thinking about ways of implementing a quasar-OTP system (section 4.4.1), I had an idea of using **analogue** circuits to perform OTP encryption/decryption. This is why I took the original SIGSALY as a starting point because it used **analogue media** (turntables) to store OTPs, and many analogue electronic circuits to perform crypto operations. With a combination of modern analogue and digital electronics, the SIGSALY-2 may cost some 200 EUR in parts - Alice and Bob can afford it!

The main idea is to use **analogue addition** (a simple adder implemented with one operational amplifier) of an analogue voice signal and analogue white noise (actually random numbers sequences “played” from an SD card) to encrypt the speech signal by Alice. After the reception, Bob will use **analogue subtraction**, i.e. subtract analogue white noise from the encrypted signal to get Alice's plain voice. The amplitude of OTP-white noise must be at least 10x amplitude of Alice's plain voice input to make her voice completely inaudible (at 1/1 ratio, Eve will still recognize Alice's “noisy” speech without much effort).

Why use this approach? Analogue signal processing doesn't require much computing power - that trustworthy old Z80 can still do the job. **The main reason**, however, is that this system can be designed to work for quasar OTP (or many similar setups, which don't require Alice and Bob to meet to exchange OTP keys!). Remember the main problem, analysed in 4.4.1? Alice and Bob can capture the same signal from the same quasar (to generate OTPs), but it probably won't be the same amplitude and there will always be some delay (in order of 20-40ms if they are both located on planet Earth, which we can assume). Even if the amplitude difference and time delay are perfectly compensated, the two signals still won't be 100% identical. This is because of the always present minor differences between analogue filters parameters in Alice's and Bob's quasar radio signal receivers, and also different atmospheric paths of quasar signals. Using OTPs in a usual “digital” way (like in OTP Crypto Shield, section 4.4) requires 100% identical OTPs (for both Alice and Bob), precisely synchronised, which is not a problem when digitally copying the OTPs e.g. from Alice's SD card to Bob's.

Compensating Alice's and Bob's OTPs in SIGSALY-2 for amplitude is relatively easy, and a bit more difficult for the time delay. Please note that this is the equivalent of searching for a lost key pointer on the OTP Crypto Shield. As we will see later, it will work much quicker combined with analogue electronics on the SIGSALY-2 system. Even if Alice's and Bob's OTPs (after compensation at Bob's receiver) are not 100% identical, this **will manifest only as some additional noise** after decryption at Bob's receiver, but Alice's voice will still be legible! The system will still work OK if Alice transmits an FSK modem signal instead of her voice since no digital voice compression is used. Let's go through it step-by-step, to set up a good Matlab simulation that worked fine for me.

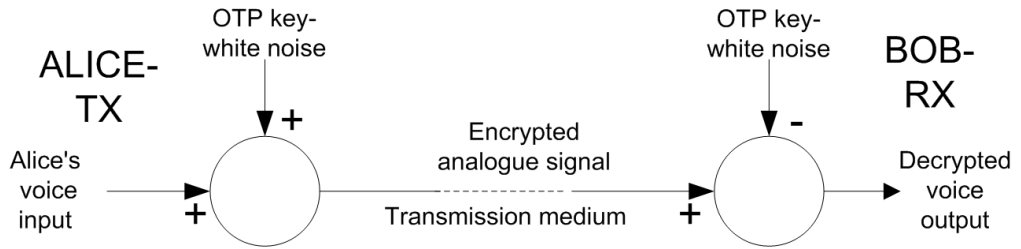


Figure 7.1 - The basic principle of analogue OTP voice encryption

Figure 7.1 shows the basic principle. Quite a few problems need to be addressed: The first is a possible difference in amplitude between Alice's and Bob's OTP noise signal. This is relatively simple. Since the amplitude of OTP key-noise at Alice's transmitter is at least 10x the voice input, it means that an AGC circuit (maybe completely analogue, something like a PI controller) on Bob's receiver will simply vary the gain factor of a received encrypted signal, to equalise its amplitude to the OTP key-noise signal. If two AC signals (OTP key-noise and Alice's voice) are added, their combined effective (RMS, or root-mean-square) value can be calculated as:

$$U_{encrypted} = \sqrt{U_{noise}^2 + U_{voice}^2}$$

This means the result will be 10.05V, for noise at 10V and voice signal at 1V. This 0.05V difference can be disregarded compared to 1V of plain voice signal. So, if the amplitudes of the received encrypted signal and OTP key on Bob's side are equal, this is good enough to proceed with the time delay compensation.

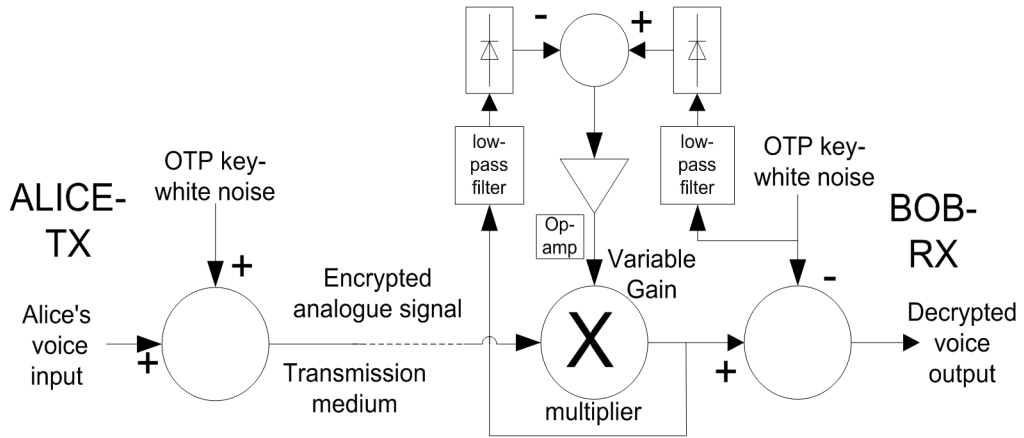


Figure 7.2 - AGC circuit to match the received signal and OTP noise amplitude

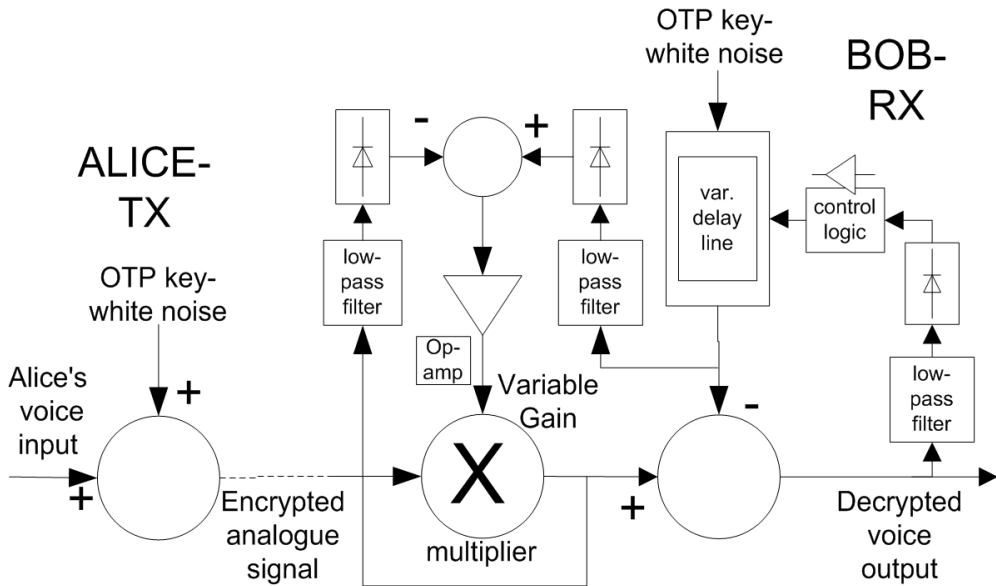


Figure 7.3 - Variable delay line - a FIFO buffer at SD card output and its control logic

To compensate for the time delay, a variable delay line is installed after OTP key output on Bob's receiver. It can be a FIFO buffer, at the output of the SD card (containing OTP keys), before converting to analogue signal. While the delay is not properly adjusted, the amplitude of the decrypted output is very high, higher than 10V (if both encrypted input and OTP key noise are at 10V), as explained before. When the delay is well adjusted, the output amplitude will drop to 1V, the amplitude of the voice signal at Alice's input because now the 10V noise is subtracted from well "aligned" 10V OTP noise and only Alice's voice remains. The amplitude of the rectified and filtered output is used by the control logic to shift the delay until it is properly "locked", which is indicated by a **significant drop of**

output amplitude. This is yet another reason why it is necessary to have a large, 10x ratio between OTP-noise and plain voice audio.

If the digital part of the system uses a sampling frequency of 8kHz (enough for a human voice, whose analogue bandwidth is up to 4kHz), this means the variable delay line (a digital FIFO buffer) needs to be dynamically adjusted and clocked in time steps smaller than the digital audio sampling time (125µs at 8kHz) - one-tenth (12.5µs) is enough. To compensate a possible maximum 50ms delay, with 8-bit OTP keys and 8-bit audio samples, the FIFO buffer needs to have a capacity of 4KB, which is not very large.

The Matlab Simulink simulation model in figure 7.4 implements all the functions defined on the chart in figure 7.3. The transport medium always introduces time delay and noises, so this is implemented in the model as well. A plain analogue PI controller with filters and rectifiers works well as AGC, to control input to multiplier for a variable gain. A variable time delay compensation line is controlled with a more complex circuit. A logic to change a “search” direction, another **PI controller**¹ can quickly control the compensation delay well, but the sign-direction can be wrong, so a multiplication with -1 (opposite direction) is used if the maximum delay time is reached. A Stop-relay stops the search if the output amplitude is low enough, so the delay time is thus fixed.

1 The Proportional-Integral controller from automation theory, is a controller combining proportional gain and integration on a control error (a difference between a desired setpoint and an actual process variable value, a measured feedback) to produce a control signal to stabilise the controlled process and thus make the feedback follow the setpoint, down to zero control error (P controller can’t achieve zero). In the aforementioned AGC circuit, the setpoint is Bob’s OTP key-white noise amplitude, and the feedback is the amplitude of the received encrypted audio signal, after variable gain multiplier. Op-amp output is the control signal (input to the multiplier). An analogue PI controller can be simply implemented with a single operational amplifier and a few resistors and capacitors.

The Matlab simulations showed good results. After Alice initiates the transmission, Bob's controllers need less than 0.5 seconds to lock to full compensation, so Alice's voice can be heard.

There is one more problem to solve. If the delay time is once fixed, it doesn't mean that it won't have to be re-adjusted a few seconds later. If Alice's and Bob's MCUs are clocked from plain XTAL oscillators, it is easy to work out that after only a few seconds of Alice's transmission, the variable delay line will have to be re-adjusted. Let's suppose that their XTALs run at 10.000MHz. XTAL oscillators not compensated against voltage and temperature variations can be expected to have $\pm 10\text{ppm}$ frequency deviation. This means it will take only 1.25 seconds for a difference (an extra time delay) of $12.5\mu\text{s}$ (one step of variable time delay FIFO buffer) to emerge between them. AGC is linear and rapid and active constantly, but variable time delay control is not so fast, so we would prefer to adjust it 0.5s at the beginning of each transmission and fix it until the end of transmission. This can be solved in a way to use e.g. GPS-controlled XTAL oscillators (both on Alice's and Bob's side), whose frequency can be stabilized below $\pm 1\text{ppb}$, locking onto very stable and precise reference signals transmitted from GPS satellites. It would take 3.5 hours of continuous transmission to generate only $12.5\mu\text{s}$ skew. GPS receiver-controlled XTAL oscillators -clocks are not expensive and can be bought for less than 50EUR. They don't transmit strong signals and don't require big antennas. Alice purchasing and installing them won't raise any suspicion for Eve. One of the reasons the original SIGSALY was so complex and expensive was its **turntables!** The speeds of rotation on Alice's and Bob's sides had to be precisely controlled for the system to work, with the technology available in the 1940s.

It is easy to see that Alice and Bob can build SIGSALY-2 for less than 200EUR. Compare it to the original SIGSALY. Apart from usual components like MCUs, SD cards, operational amplifiers, etc, some more specialised components like analogue multipliers and FIFO buffers will be required, but these are also easy to obtain.

One of my colleagues remarked that comparing this system to the original SIGSALY doesn't make much sense, because they are very different. He is right. Their in-common traits are that they are both audio/voice encryption devices, that both use one-time pad encryption, and a combination of analogue and digital circuits to encrypt/decrypt the audio. The original SIGSALY was designed to compress speech down to some 1500 bps, to make a baseband as narrow as possible, to reduce the RF power required for long-range shortwave transmission without any repeaters in between. It used a complex combination of time-domain and frequency-domain procedures to process audio signals, and it transmitted digital encrypted information in modulated HF radio waves. SIGSALY-2 uses a standard 3-4kHz speech analogue audio range signal to modulate RF and carry encrypted information and does everything using simple time-domain procedures. At the sampling frequency of 8kHz and 8-bit samples, this would be an equivalent of 64kbps (compared to 1.5kbps at original SIGSALY), and would consequently require much more RF power for long-range shortwave transmission. I didn't take it as important, simply because a 4kHz long-range audio transmission is not a problem for Alice and Bob. A lossless VOIP internet link is easy to set up. SIGSALY-2 was designed to be cheap and simple and built using easily obtainable components, to still be able to implement virtually unbreakable OTP in combination with

fast synchronisation and tolerance to a slight mismatch between Alice's and Bob's OTP keys (which will inevitably happen when generating OTPs from radio signals received from quasars or encrypted satellites).

I named it SIGSALY-2 because it does an equally good job for Alice and Bob as its original predecessor did for the government: protecting secret info with unbreakable OTP, for much less money.

7.2 • Microwave oven - an innocuous machine?

It's time for some microwave electronics again. As I already showed many times in this book, they are extremely important for Alice and Bob. Microwaves make critical information leaks through tiny holes (which makes these leaks difficult to detect) and may also interfere with the operation of other electronic devices, making them malfunction, apparently without logical cause.

Microwave ovens have been around with us for a while. They operate by generating microwaves, typically around 2.45GHz (consumer-grade ovens), or lower UHF frequency (high-power industrial types). The typical power of a household kitchen microwave oven is around 1000W. Their microwave radiation is generated by **magnetrons**. They are one type of **cavity resonator**, which are used as RF oscillators in the microwave range (above 1GHz). Remember **helical resonators**, explained in 2.4.1, which made van Eck's phreaking possible? They are used for frequencies up to 1GHz. Above this, the dimensions and number of turns of inductor coils become very small and few, so the "coil" boils down to one simple short straight conductor, without any noticeable windings. A cavity resonator is derived from a helical resonator, with the coil "degenerating" to one short wire as the microwave frequency increases. The oscillating frequency is determined solely **by the physical dimensions of the cavity**, the same as the dimensions of helical resonator box walls determine its resonant frequency. The same principle is applied to design e.g. **microstrip microwave filters** (also explained in 2.4.1). They have no distinguishable capacitors or inductors, only microstrips of certain dimensions, which function as space-distributed capacitances-inductances (in one element), with all the effects of **transmission lines** since their dimensions are comparable to wavelengths of the RF signals used.

Apart from resonating at a certain frequency, cavity resonators also produce significant RF power, so don't need additional power electronics to amplify the power of an oscillator's output. A magnetron is a type of cavity resonator that uses a magnetic field to direct a stream of electrons, which start to oscillate as they pass along several cavities. RF power is taken from the magnetron resonator through a waveguide to the oven's heating chamber. A magnetron's frequency can't be precisely controlled, which is not important for microwave heating.

In the ideal world, all RF power would stay contained inside the heating chamber. Since these are consumer-grade products, it can be reasonably expected that less than 0.1% (or 1W) of 1000W radiated by the magnetron will escape. The frequency is very close to standard Wi-Fi channels (spaced around 2.4GHz), so small RF leakage will not immediately

attract a great deal of attention.

We looked at how Eve managed to use a regular microwave oven as an in-plain-sight listening device in the spy story at the beginning of chapter 6. Similar to the bugging [41] that took place in the USA embassy in the Soviet Union which started in 1945 and wasn't discovered until 1952. "**The Thing**" was a passive type cavity resonator without any electrical power source or active electronic circuits involved. Just a cavity chamber with a certain capacitance and inductance, using a 23cm straight wire as an antenna (the frequency used was around 330MHz, so this was a $\frac{1}{4}$ wavelength stick). One wall of the cavity chamber was a **thin membrane, acting as a microphone**. Being a completely maintenance-free, fully passive device without batteries, it needed to be "illuminated" by a 330MHz radio transmitter from across the street. Motion of microphone membrane (picking various audio signals from ambassador's office) cavity wall would change the resonant frequency of the cavity and modulate the received RF, also re-transmitting the modulated RF through its antenna.

In the microwave oven setup, no external RF transmitter is required (its strong signal can be detected) since the oven already has a 1000W microwave "transmitter" installed "legally". If only 0.1% of its power escapes from the oven, it is still much more than the RF power of an average WiFi hotspot, to be picked at even greater distances. Eve perhaps has to tamper with magnetron walls (to install **thin aluminium sheets** to act as **microphone membranes**). Since the RF resonant frequency is affected by dimensions, maybe even this is not required. Even the dimensions of the waveguide and oven's door can slightly affect the magnetron's frequency (magnetrons don't have a highly stable frequency anyway, by their design). The oven door surface can also vibrate at audio frequencies. In the end, you get a combination of AM and FM, with a carrier at 2.45GHz, modulated by kitchen audio signals.

Eve could tamper with waveguide flanges and seals to increase the amount of RF energy escaping outside if needed, mainly because the **modulation index**² can't be expected to be very high in this setup. This still wouldn't raise much suspicion in the year 2000, when lots of cheap, highly radiating consumer electronic devices were already around.

As if this was not enough, Mallory had also decided to step in before the oven was delivered to the embassy kitchen. A microwave oven normally has a microswitch on its door, as an interlock to disengage the magnetron when opening. In the year 2000, microwave ovens were already controlled by MCUs. She changed the MCU's firmware and programmed it to engage the magnetron (to create a short 100ms RF pulse at full power), cca. **one second after the door is opened**. She targeted one specific person, a high-level official with a heart condition who had a pacemaker. Pacemakers are electronic devices, implanted in the human chest cavity, which control heartbeat pace by sending electrical signals to the heart muscle. They operate on microwatts of power and don't respond well to 1000W microwave RF bursts less than a meter away. So, his pacemaker was zapped (without any visible

² modulation index: a percentage (a ratio to a possible maximum), telling how much the modulated variable (amplitude at AM or frequency at FM) of the carrier wave signal varies around its un-modulated level

traces left), and his death was attributed to his bad heart condition and/or pacemaker's accidental failure. The oven wasn't suspected, since it had been previously inspected. Needless to say, the technician was fooled and thrown off by a "warning" which came from Mallory! Looking specifically for a "**covert listening device planted inside**", he was expecting to find some extra installed electronic circuit, but he couldn't find anything, because it wasn't there. This is why Eve's tampering with the waveguide elements (flanges, seals, and thinner walls) passed unnoticed. It was even more difficult to detect Mallory's tampering with the MCU's firmware, installed with a completely different purpose.



Figure 7.5 - An innocuous, but highly versatile kitchen appliance

Since the magnetron is turned on and off through a high-voltage transistor (not an electromechanical relay), the brief 100ms burst of RF power when opening the door couldn't be easily noticed. He was thinking too much with 20th-century³ mindset, disregarding the fact that it is usually **software, not hardware** modification that subverts your machine against you in the 21st century. On the other hand, he was too quick and sloppy when making the inspection, which is unfortunately very typical for 21st century.

Transistors are used to control the power of magnetrons (in the 0-100% range), by changing the PWM duty cycle. Turning a magnetron on and off in milliseconds is not a problem, since their amplitude stabilisation when switched on/off is a matter of microseconds. This wouldn't be possible with electromechanical relays.

Now I must emphasise the importance of "hardwire programming" for both safety and security again. I have mentioned it many times in this book already. If the door switch

3 Pardon me, the year 2000 actually belongs to the 20th, and not the 21st century. The staged "Y2k" frenzy was a complete fraud, starting from its incorrect name in the first place - "the millennium bug". A worldwide computer breakdown was supposed to happen on 1st of January 2000, and "year 2-kilo" still belongs to previous, 2nd millennium by all the time-keeping and counting standards. Millions were wasted on those ludicrous "Y2k certificates".

interlock was hardwired to magnetron power circuits (or to an electromechanical relay which would cut the power), the attack on a pacemaker by tampering with MCU firmware wouldn't be possible. Hardwiring usually seems expensive and old-fashioned (just bring all the signals to MCU, and let its software handle everything), while its positive effects on security are routinely disregarded.

Since the MCU can modulate the power radiated by a magnetron, as a part of its normal procedures to control the heating power (like on any other oven), the PWM control of the power transistor (controlled by MCU) can also be subverted by Eve! She can make it modulate the 2.45GHz RF much faster than the normal heat-power control PWM, while still maintaining the regulated heating power at the desired level. Since magnetrons can be modulated in microseconds, she can use this approach to modulate secret data gathered over long periods into the 2.45GHz carrier at high speed, transmitted each time at a fast bit rate when the oven is powered up! Sensors used to gather the data (like microphones to pick conversations inside the kitchen while the magnetron is not powered on) are another problem, of course.

Installing the listening device in a kitchen may not seem like a good idea, but I am amazed by the amounts of confidential information, critical for a company (any company, not necessarily a 3-letter government security agency, I have no first-person experience in that sector), which regularly leaks out during coffee breaks, or chit-chat gossip sessions, whatever you prefer to call them.

7.3 • “Funcard” system for secure digital signing and decryption

We will now take a short break from microwave electronics, until the next section. Now we will say a few words about the security of any asymmetric public-key cryptosystem and ways to improve it. This may apply to message encryption/digital signing software like PGP, as well as Bitcoin [42] (or any other crypto-currency) which is also based on pairs of public and private keys. **Both PGP and crypto-currencies may in this day and age be very important for Alice and Bob!**

In section 4.2.2, we analysed the RSA encryption method (used in PGP software), and its problems. Bitcoin uses elliptic curves⁴ instead of multiplication of big prime numbers (RSA), but the principle is the same. Everything is based on pairs of public and private keys. Now let's go through a list of possible problems (already mentioned in 4.2.2) that may arise in any similar crypto-system:

1. Alice may not be able to reliably oversee the process of public-private key-pair generation, and protect it from an eavesdropping Eve.
2. Alice may not be able to reliably oversee the processes of encryption/decryption and signing/verification.

⁴ elliptic curves: functions based on equation $y^2 = x^3 + ax + b$, defined over limited, or finite fields of integer numbers-based on modular arithmetic, using mod (integer division remainder) operation to limit the field of integer numbers between $[0, d-1]$, where d is divisor, similar to PRNGs previously described in section 3.2.1

3. Alice needs to keep her private keys hidden from the rest of the world.

There is no perfect way to solve any one of these three problems, especially if working on a general-purpose computer or a smartphone. Programs like PGP or any **crypto-currency wallet**⁵ usually solve problem 3.) by encrypting the private keys with Alice's special password-passphrase and keeping them encrypted (on local disk or even on webserver), revealing them only when decrypting or signing a message.

PGP is usually run on Alice's computer, but crypto-currency wallets run even in online-web implementations which are very insecure. Here I will outline a simple wallet, with an emphasis is on **protecting the secrecy of private keys**, trying to fully solve problem 3.) and also the other two as well.

As we have already established, crypto-currency payment systems have good long-term prospects. Since most wallets have many security pitfalls, they are worth improving. When Alice sends crypto-money to Bob, she needs to sign (or authorise) a Bitcoin transaction (figure 7.6). Alice's wallet contains several **bitcoin addresses**, each carrying a balance (figure 7.7). A Bitcoin address is an equivalent of a **bank account number**. It is public information. Knowing Alice's bank account number enables you to send money to her, but not to spend her money! Please note that getting a single bank account number is a relatively complex procedure while getting multiple bitcoin addresses is just a matter of generating random numbers by a bitcoin wallet. Some wallets don't even check the random-generated addresses online for **collisions**, although it is theoretically possible for two wallets in different parts of the world to generate the same bitcoin address. Such an event is extremely unlikely to happen.

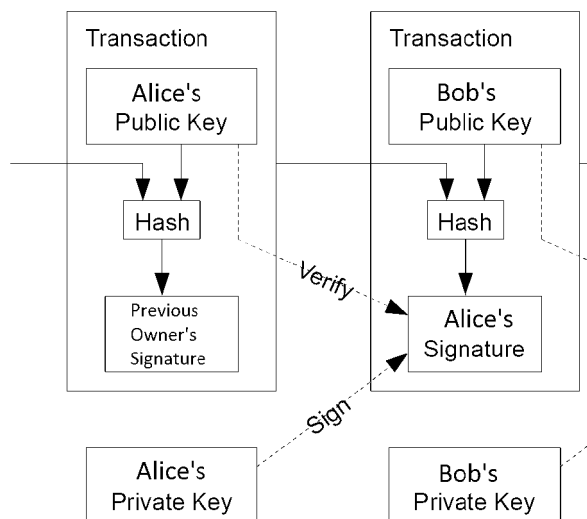


Figure 7.6 - Digital signatures in blockchain transactions

⁵ crypto-currency wallet: a software (or even hardware), designed to handle crypto currency transactions

Type	Address	Balance	Tx	Label
receiving	1PXknTonLbU4cNuHhs48BueAcbcEfGGdU	0.	2	
receiving	1Lx1Qt9WjYHURgeboZvMroU3A6GpgrFD8h	0.	2	
receiving	1NahYkzdmKUXJPYwY5WQrxLBccvRxhREo	0.	2	
receiving	19hk38C6nV3AqgjRVCYkTUAoQSDUnxgi7v	0.	0	
receiving	13xXcEk6mynJNkaRge6qg71zpUnS9bPF6r	0.	0	
receiving	17yPeJeRNmg5cmEKDLXiXG3BSKw3LXHkmk	0.	2	
receiving	1MgX9eua6uh5ynwyQmUYXwZwkPkdMTy3jm	0.	0	
receiving	19FyqPnkxwZrRUwmNyEriTuc8VRHX8T1D	0.	0	
receiving	17CeTHknNQZqnYExYbxG2vLGqzJsWgG74N	0.	0	
receiving	1CffYEgG6mt8o797JiGamNJ88rbKn4Hdue	0.	0	
receiving	14HBt4H3sTW9KneBZT6azB3SyHXK8qcyfA	0.	0	
receiving	1LFCw6vABJUGzobMtfCKHtF3Ys5o8fPd	0.	0	
receiving	1MQ4Gg7cLj89nTzr5aVujCzHUcpNEodMo7	0.	0	
receiving	16xLR2zEMawZPkf7j7bKT97bSjjNc8VDGU	0.	0	
receiving	1DbQfuR9ekRaSHEcHTctwfdCTG1hHYGHF5	0.	0	
receiving	1EakoRHhH56nQ8UKcZJFHYudx61n9BwVfN	0.	0	
receiving	16VVMfTqFtbyggVFUBghPibsdaf5yH5Vht	0.	0	
receiving	1B9KqW6J3D7A72jdQB2gvtbxmq8274SmZ4	0.	0	

Balance: 0.0 mBTC

Figure 7.7 - “Electrum” Bitcoin wallet screenshot

A wallet generates multiple pairs of public and private keys from random numbers and then calculates bitcoin addresses from public keys. It is done by processing a public key with a few hash functions, and a conversion to **base58** format (explained in 4.2.3) in the end. A bitcoin address is a shorter, compressed variant of a public key. Money is hence transferred from Alice’s bitcoin address to Bob’s bitcoin address (both closely related to a pair of public and private keys, one pair for each address), the same as a transfer from Alice’s bank account to Bob’s.

A chain of transactions (figure 7.6) is kept in a public database - ledger - which is called a “blockchain”. When Alice wants to transfer an amount to Bob, it is started by generating a hash (on the right-hand side of figure 7.6). It is calculated from data based on Alice’s balance at her bitcoin address, her public key and previous transaction, and Bob’s public key. Alice signs this hash with her private key (this is a security-critical part because the secret private key here needs to be revealed in plaintext form for the signing operation to be executed). Bob then verifies the signature of the hash with Alice’s public key and accepts the transaction if it checks. Now he needs to wait for confirmation of transactions in the

blockchain, performed by bitcoin “**miners**”, to prevent a possible double-spending.

Hiding the private keys is critical (and not adequately implemented) for any kind of bitcoin wallet. Let’s start with an analysis of the types of bitcoin wallets in use today (figure 7.8).

- **Online wallets** (the top of the picture) are the worst from a security standpoint. You need to trust a server, connected online 24/7 to keep and guard your private keys.
- **Desktop computer (or laptop) wallets** are much better because they store private keys locally. These private keys can still be stolen (during the short time of signing/decrypting a message, when they are in plaintext form). Their encrypted versions can also be cracked. They can be copied while being generated from random numbers, or even re-generated if the RNG is weak (please note Alice has no information about the RNG in a general case).
- **Mobile wallets** are practically the same as desktop wallets, they run on smartphones, and they are consequently much less secure in all ways than desktop wallets.
- A **paper wallet** is just a piece of paper with a pair of public and private keys written on it. Both are usually encoded using QR-codes. They can’t be electronically hacked (like all other electronic wallets). On the other hand, the private key is written in plaintext, so needs to be read by an electronic device when sending money to someone - an insecure or suspicious electronic device (like e.g. a bitcoin ATM) thus must be involved at some point in time, revealing the private key plaintext form for even longer than a desktop wallet.

The system that I will propose is a **hardware wallet**. The problem with standard hardware wallets is that they are usually USB (like in figure 7.8) highly integrated black boxes, so Alice can have no control or supervision over them. The good thing is that they can **keep the private keys and sign transactions themselves**, so the private keys will always stay inside them, only hashes coming into input, and signatures going out on output from them.

With this in mind, let’s try designing a simple hardware device that can be built using DIY techniques from simple general-purpose components and hence is supervised better than a standard USB hardware wallet. Let’s use a general-purpose smartcard (some easily available types have been around for more than 25 years, produced in their millions) to keep private keys and sign transactions. A simple circuit based on an MCU and USB->UART converter will handle and supervise the data traffic between a PC (or a smartphone or any insecure device) and the smartcard.

CRYPTO-CURRENCY WALLETS

ONLINE



Online cryptocurrency wallets are online digital wallets which can be accessed online through your internet browser.

PROS:

- Easy to set up
- Easy to access
- Free

CONS:

- Fake websites are endlessly trying to phish your private key.

DESKTOP



A desktop cryptocurrency wallet is a software which you download to your desktop and can access offline.

PROS:

- Easy to set up
- Easy to access
- Safer for online purchases

CONS:

- Regular updates
- Not practical to carry around.
- Not a longterm solution

MOBILE



A Cryptocurrency Mobile Wallet is a digital wallet installed on your smartphone.

PROS:

- Easy to install
- Easy to set up
- Easy to carry
- Easy to make physical purchases

CONS:

- Private keys get lost very -
- App can get deleted accidentally

HARDWARE



A cryptocurrency hardware wallet is a pendrive-like wallet which stores your private and public key. The hardware wallet extremely versatile because they are easy to carry.

PROS:

- Offline
- Easy to carry
- Multipurpose

CONS:

- More expensive
- Not all models are user friendly

PAPER



A cryptocurrency paper wallet is the private key and public key written or printed on paper.

PROS:

- Offline
- A great back up system from your wallets other wallets

CONS:

- Gets damaged over time
- Easy to lose a

Figure 7.8 - Various types of crypto-currency wallets

Besides the article [23], already mentioned in section 4.2.5, you should also read [43] to become well acquainted with smartcards. Although there are some other types, like memory-only cards (containing only EEPROM memory, but no CPUs), or rather complex cards with operating systems and BASIC interpreters (like BasicCard by ZeitControl [44]), most of them are just **PIC or Atmel-AVR 8-bit MCUs** with **extra AT24C EEPROM** memory added. I chose **Funcard** (also called “**Purplecard**”, see figures 7.9 and 7.10) for several of my projects for many reasons (besides its amazing design) that I will now explain. Although initially designed for cracking encrypted (or scrambled) TV signals, these are general-purpose devices that can be programmed to do almost anything.

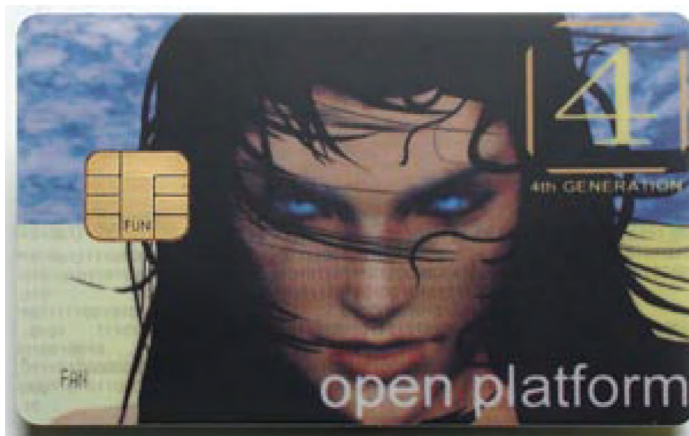


Figure 7.9 - A standard, 4th generation Funcard (with AT24C256 EEPROM)

Not only is it based on Atmel AVR that I am familiar with but its interface to card connector (both RFU pins used) and assembler instruction set (AT90S8515-Atmel AVR) are implemented in a way that enhances security. Unlike some other MCUs, Atmel AVR's are programmed using the SPI protocol - which requires 3 lines (MISO, MOSI, and SCK). Although a standard smartcard connector has 8 pads (Vcc, GND, RST, Vpp, CLK, I/O, and **2xRFU**—“**Reserved for Future Use**”), these RFU pads are seldom used (along with Vpp, which was used only on old types, for 12-14V programming, like on old UV EPROMs). None of the credit cards I have ever tested used these RFU pads, and the same goes for PIC-based Smartcards (like Goldcard or Silvercard) and also ZeitControl BasicCard. This effectively means it is not possible to re-program a Funcard without connecting the RFU pins (MOSI and SCK signals of the SPI protocol). This is very important from the security standpoint - if the ISP2 connector (figure 7.11) is not connected, the Funcard firmware **can be neither written nor read!** Furthermore, since the AT90S8515 **doesn't support the spm** (store program memory) instruction (analysed in section 2.3.2), this makes a Funcard even more appropriate for this application. Again, without the programmer connected to ISP2, the Funcard firmware can't be read or written. All the other aforementioned cards are programmed through bidirectional I/O pin, otherwise also used for regular run-time data input/output, which is very bad from a security standpoint. Low-tech rules again.

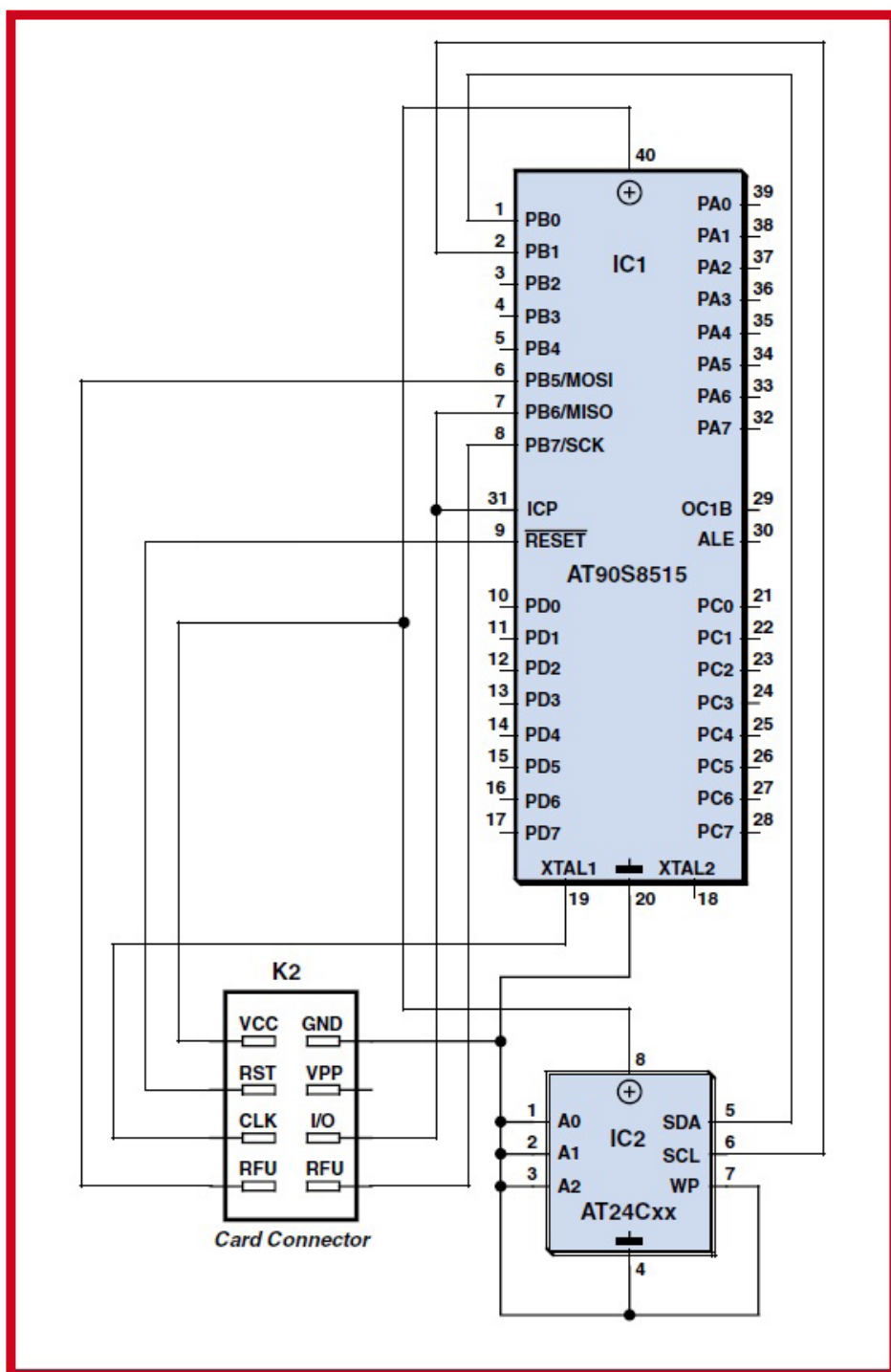


Figure 7.10 - The internal circuit of a Funcard

The rest of the circuit in figure 7.11 is very simple. A USB-UART converter handles communication with wallet software running on a PC or smartphone. This wallet software for this system is programmed in a way that it stores public keys (and hence the bitcoin addresses), **but not private keys** - they will be stored on the Funcard only! The PC wallet software handles everything else like a usual desktop wallet, except for signing the transactions. The hash and public key/addresses are sent to the Funcard, and then its firmware signs the transaction and returns the signature to the PC. The “Supervisor” MCU monitors and controls the data going to the Funcard. The MCU will pass the message to the Funcard if it is well-formatted. It will not pass the message if a Trojan on a PC tries a **fault injection** attack (performed by sending badly formatted data). For example, some private-key algorithms can reveal the private key in case of a short all-zero input (0x00000000...). It can also monitor the Vcc voltage and halt the system if it drops too low (like below 4.0V). This indicates another type of fault injection attack, a so-called **“voltage glitch”**. Two separate ISP connectors are used to program the ATmega8 and Funcard.

The Funcard will be treated with more care than a credit card and guarded very carefully by Alice (it can’t be insured against theft like a balance on a credit card). Private-public key pairs will be generated by a **“clean” computer that never connects to the internet** (they are created based on random numbers without a practical need to check anything online) and loaded to the Funcard. The Funcard will then send the public keys to the PC wallet software (running on a general-purpose PC connected to the internet). The aforementioned “clean” computer can be specially designed hardware, equipped with a good TRNG for generating truly random key pairs. This will keep them both random and truly secret.

The Funcard and its hardware wallet on figure 7.11 could also be replaced with a Z80 system. This is not very practical to carry around like a Funcard, but more secure if Alice suspects her Funcard might have been tampered with. Please note that signing a transaction isn’t very computationally demanding, and it is not important if it takes several minutes on a Funcard, since a bitcoin transaction almost always takes more than an hour to fully complete.

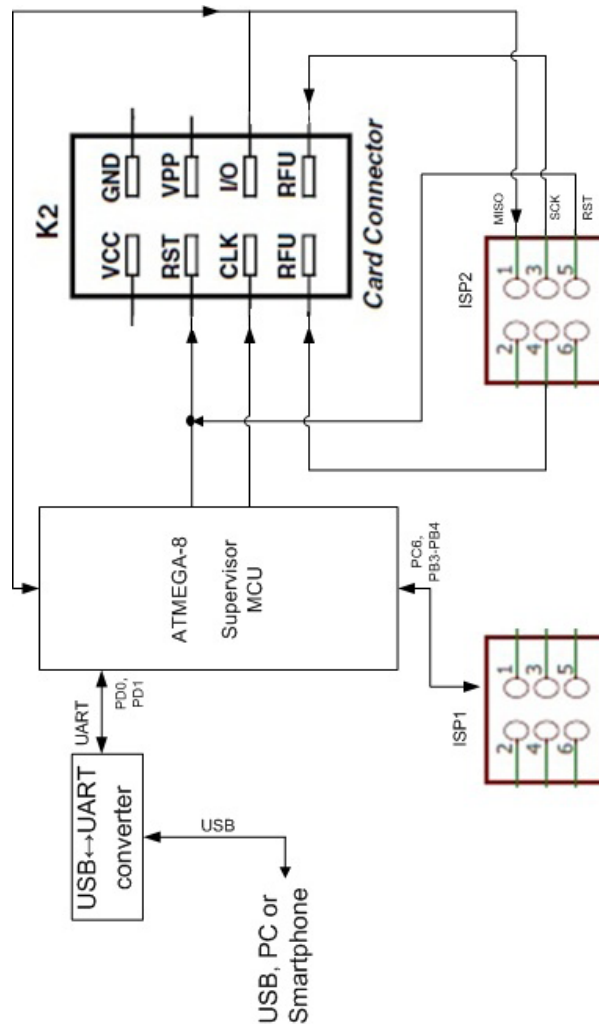


Figure 7.11 - Principle schematics of hardware of a Funcard wallet

Besides handling private keys for cryptocurrency transactions, the same hardware can be used for PGP message encryption software as well, to decrypt and sign messages. It will work as long as the secret messages are reasonably short, which they usually are. The same as the special bitcoin wallet software required, this will also require a specially programmed version of PGP software, which doesn't generate the keys, and doesn't store private keys on a PC. Since it will be used not only for signing but also for decrypting the secret messages, it is best to modify the hardware to send the decrypted plaintext directly from the Funcard to for example a thermal printer, to avoid sending it to a general-purpose PC or smartphone.

In the following chapter we will talk more about this and how to design a special terminal, to keep sensitive secret plaintext away from insecure networked PCs and smartphones.

7.4 • TEMPEST-proof terminal

Now it is time to outline a design of a **dumb terminal** for secret communications. It must be TEMPEST-resistant as much as possible and as dumb and simple enough that you can trust it. Sometimes it seems that getting a smartphone nowadays is much easier than a **dumb phone** [45].

Section 2.4.2 contains a list of guidelines to be followed to defend against TEMPEST on a DIY budget. Let's start with an aluminium box, the size of an old A4 format typewriter. A keyboard will be the input device, and a thermal printer will serve as an output, with 57mm paper tape, coming out through a 60x0.5mm slot, at least 10cm long to increase attenuation for all RF frequencies below 2.5GHz (higher frequencies are not likely to occur). A simple MCU or small Z80 computer will handle the scanning of the keyboard and communication with the printer and crypto-devices. The box should have enough extra space for a TRNG and various crypto-devices (including SD card copiers) to be put inside in a modular manner as required.

Let's calculate a penetration depth in aluminium (using good-conductor formulas, like in section 2.4.1), for a frequency of $f=100\text{kHz}$. TEMPEST radiation at lower frequencies is unlikely to create sufficient RF radiation at short lengths. For $\sigma=3\times10^7\text{S/m}$ (conductivity of aluminium) the penetration depth is $\delta=290\mu\text{m}$. If the box is made of aluminium 1mm plates, it will block even 100kHz adequately. Now it is clear that the keyboard will be the main problem. Its shielding is very difficult to implement. Even if the keys are somehow covered with heavy-duty 50 μm thick aluminium foil, it can be expected to adequately block only RF frequencies above 10MHz, since at $f=10\text{MHz}$, $\delta=29\mu\text{m}$. This is not good enough, although most RF TEMPEST leakage in practical scenarios happens at higher frequencies (it happened between 80MHz and 120MHz on the keyboards that I tested in chapter 6).

Perhaps a solution is not to scan the keyboard but use 100 thin wires to connect each key separately. A flat ribbon 100-wire connector would be used to connect this specially designed keyboard to the box. Every key then behaves as a low-frequency switch. No keyboard scanner-controller is then needed. This, however, doesn't seem like a practical solution, since each wire would need to be filtered to prevent leakage (like a 2-stage lumped + distributed filter needed to filter the power supply, as analysed in section 2.4.2).

It is better to install the keyboard in the aluminium box than outside of it to reduce the RF leakage. Maybe if keys are made of aluminium and spaced very closely, with less than 0.5mm gap between them, this could reduce the RF leakage below detectable level, especially if the whole keyboard is additionally covered with aluminium foil. Since the lengths can be expected to be longer than 60mm (width of a paper tape slot), the aluminium foil cover is a must, since these slots can then be expected to radiate RF even in the sub-GHz range.

Speaking about the power supply, it is better to use classic iron-plate laminated core transformers operating on 50/60Hz mains frequency than switch-mode buck converters operating on much higher frequencies. They are more likely to leak MHz frequencies to mains power supply cable than iron-plate cores designed to work on 50/60Hz. Battery power may still be considered more practical, eliminating the need for a mains cable.



Figure 7.12 - Amphenol connectors, male (on a cable) and female (on a bulkhead)

All cable connections to the box should be made using shielded cables and Amphenol-type connectors, as shown in figure 7.12.

After the box is assembled and tested for functionality, the RF radiation must be measured in a wide span (like 100kHz - 3GHz) using a good spectrum analyser. If some leakages are still detected at several RF frequencies, these can be solved by adding properly designed jammers to block the frequency spans. A good TRNG is a must even for this, so it is better to install one permanently inside the box, to perform all of its tasks (including generating one-time pads) there.

7.5 • False Morse signature generator

Now it's time to address some pitfalls of low-tech. As already mentioned at the end of section 2.4.2, low-tech is almost always more secure than hi-tech, but **not without exception**. Hiding your secret encrypted message in megabits/second of internet traffic while connected to a WiFi hotspot in a public place (like a coffee bar or a train station), wearing a full-face anti-Covid ski-mask may be less suspicious and conspicuous than erecting a shortwave antenna and transmitting a Morse telegraph message.

This section is about concealing a telegraph operator's identity. If Alice simply uses a keyboard to enter plain ASCII ciphertext characters, letting the MCU-computer convert them to dashes and dots, a "signature" of her hand (and her Morse key as well!) will not be embedded into the transmission. Her identity will thus be concealed, but Eve will know that signals were generated by a computer. All dashes and dots will have a perfect ratio, and there will be no button-bounce or ring-out of the Morse key.

The idea behind this is to make a transmission sound like it was generated manually using a Morse key, **but by another human operator**, by artificially generating non-perfect dash/dot timing ratios, along with button-bounce. This may confuse Eve. A good TRNG will be used to make the timing periods fluctuate (a human operator can never reach perfect timing as a computer). This is almost like falsifying someone else's hand signature.

There are two different types of timing that need to be taken into account.

1. The first is in the order of 100ms and longer, and these are the periods of dots, dashes, and pauses between them, that have distinct ratios for every telegraph operator.
2. The second type of timing is in order of milliseconds, and this is oscillation caused by the button-bounce effect of a Morse key (similar to any mechanical switch). This depends on the type of Morse key used, but also on the motion dynamics of the operator's hand. Just remember how and why "Kestrel" was renamed "Tomcat".

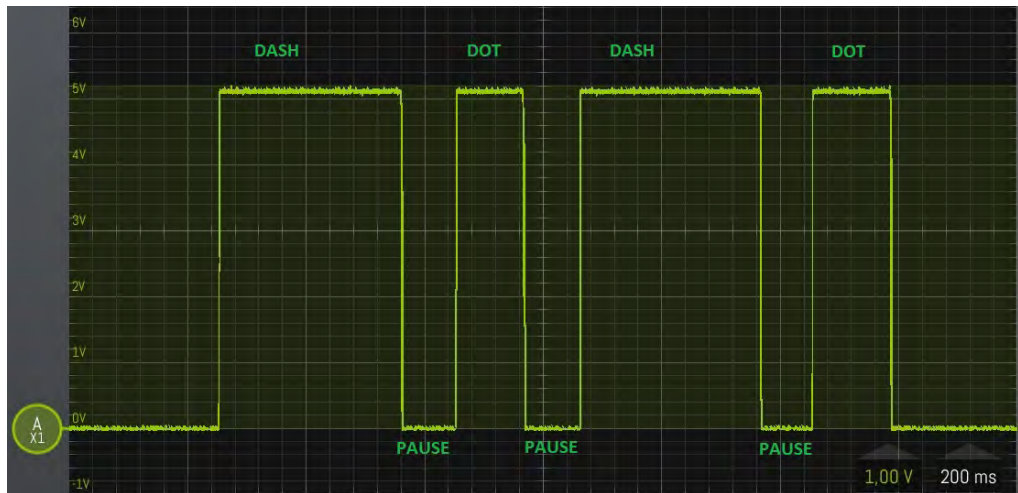


Figure 7.13 - Morse letter "C", or dah-dit-dah-dit, keyed in manually

I haven't had any serious Morse telegraph training course, so this is how precisely I can key in -.-. or the letter "C". Please note the differences between dots, depending on a position in a single letter. I have a tendency to make a dot at the end of a character a bit longer (making it a part of my "signature"). A ratio between my trailing dot and the pause before it is around 1.3:1, but still good enough to make the character legible. Now let's check the button-bounce effect.

Figures 7.14 and 7.15 show how different a button-bounce effect can be if you hit the same Morse key harder or softer. It is easier to observe when pressing the key (figure 7.14) than when releasing it (figure 7.15). Transitions in a matter of milliseconds will be modulated to RF carrier, and this will also enable Emma and Eileen to **hear** a difference (because the button transition frequencies are well within human audible range of frequencies) between a rough and a gentle hand, without even using an oscilloscope.

Designing an MCU-controlled device, assisted by a TRNG to simulate the effects displayed on figures 7.13, 7.14, and 7.15 shouldn't be difficult. Adjusting it may even be easier by listening to dit-dah sounds rather than using an oscilloscope.

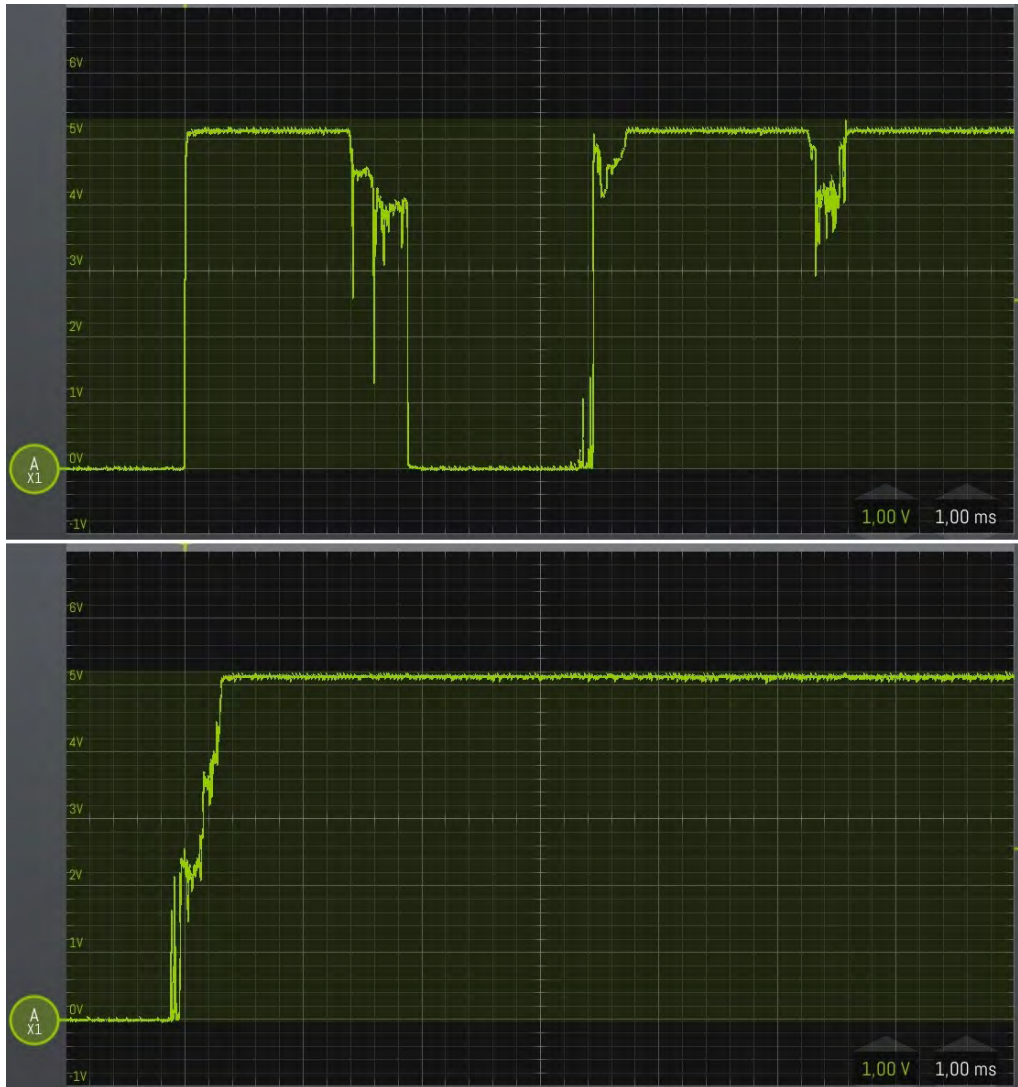


Figure 7.14 - A rough hand (up) and a gentle one (down) pressing the Morse key



Figure 7.15 - A rough hand (up) and a gentle one (down) releasing the Morse key

I once saw (sorry, I can't remember where and when) a Cold War Morse signature "spoofer" implemented with several different selectable electromechanical relays. Each of them had several adjustable screws and springs to change the dynamics of motion and oscillation of its contacts. Nowadays it can all be realised electronically. On the other hand, MCU-controlled analogue electronic outputs can simulate a much wider range of different ring-outs, some of which may be (after a careful signal analysis) recognised by Eve as artificially generated (impossible to be generated by mechanical contact, because they always have some mechanical limitations).

7.6 • Encrypted ROMs

This method was already used in the past (in the 1980s and early 1990s), but for other purposes. It was for **copy-protection** on arcade gaming machines. In the 1970s, gaming machines were being designed mainly by hardware, implementing a special hardware design used for every game. Copying them meant simply copying the circuits (designed with standard TTL or CMOS chips) and PCB designs. It was relatively easy.

In the 1980s, programming game software became important, although arcade gaming machines still had a lot of hardware-specific components for each particular game. EPROM ICs were used to store the program code, and they were also relatively easy to copy. All you needed was an EPROM programmer and a UV lamp since it was easy to read a program from a general-purpose 27C UV EPROM. Game designers started looking for methods to protect their programs from unauthorised copying. One method was to use a battery-backed RAM to hold the program code, or at least some of its crucial parts, without which the game couldn't function. The code in RAM would be lost if the battery was disconnected. A method that could be interesting for secure crypto device design was the following: UV EPROMs were loaded with encrypted code. Such code could not be executed directly on a standard CPU (usually Z80 or Motorola 68000 on gaming machines), without first decrypting it. A standard Z80 was used as a CPU, but the data it read from/wrote to the data bus would have to be decrypted/encrypted in real-time. Please note the data temporarily stored in RAM would also be encrypted with the same key in some variants of this design.

What method of encryption could have been used in the 1980s to run in real-time, but fast enough not to slow down the normal CPU operation? Using a single, constant 8-bit byte (a key) XOR-ed bitwise with a byte on a data bus (using 8 XOR gates) was a very simple, but effective method. This is similar to OTP encryption, but using a constant byte as a key all the time, so it can't be considered the same as a real one-time pad (actually it's more like Caesar's cipher). Figure 7.16 shows the basic schematics of one possible design. Please note that only 4 lower bits on the data bus are shown, for the sake of simplicity.

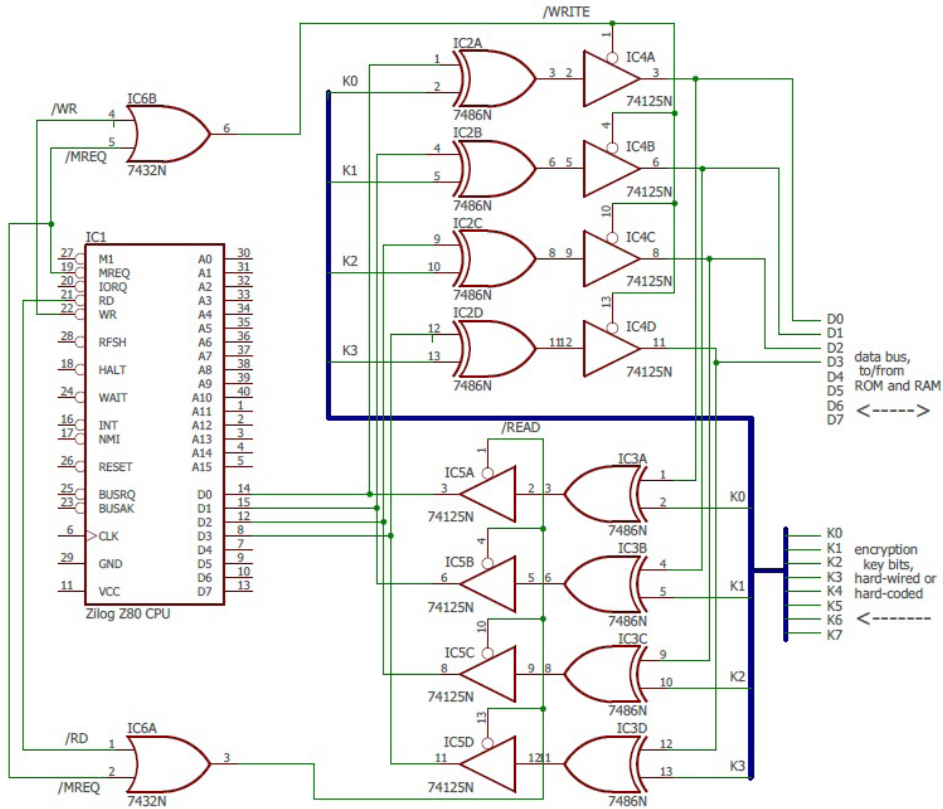


Figure 7.16 - A simple system with encrypted ROM

The data bus is bidirectional, so 16 XOR gates are needed (8 are shown). All 74xx gates on the schematics (everything except the CPU) were integrated into one IC. Let's call it "**Crypto-IC**". Eight constant bits K0-K7 define an 8-bit key, hardcoded inside the Crypto-IC. Please note other variants of this are also possible. Maybe it is better to make it work in a way to encrypt/decrypt data from ROM only, or only instruction opcodes (both for RAM and ROM), while handling data from/to RAM in plain form. Why not encrypt the address bus as well? Only 16 extra XOR gates are needed for this since the address bus is uni-directional. If three Crypto-ICs are used (one bi-directional and two uni-directional), this increases the keyspace to 24 bits. Encrypting the address bus effectively shuffles the program code all over the address space, so Eve (or a "**pirate**") now needs not only to decrypt the instructions but also to put them in the correct order of execution.

Copy-protection on video gaming machines worked like this: an 8-bit key was chosen first. Every byte of the program in ROM was then XOR-ed with this 8-bit key and stored to UV EPROM in this encrypted form. An appropriate Crypto-IC with a matching 8-bit key hardcoded inside had to be inserted into the PCB and then the system would work fine

with a normal, unmodified Z80 CPU. Although the Crypto-IC was an **ASIC**⁶, it wasn't very expensive for the gaming company to produce or purchase, since its level of integration was low, even at the beginning of the 1980s. They had to produce 256 (actually less, since some easy-to-guess keys were skipped, like 0x00 in the first place) variants of this Crypto-IC and match it with the same number of different encrypted variants of software code in the EPROM.

This worked fine against pirates trying to copy the firmware. At least this made their job a lot harder than simply copying a UV EPROM or PCB circuit. It relied on security by obscurity, because if a pirate knew how the Crypto-IC was designed and how it worked, it would be easy for him to read the 8-bit key. All he had to do was to remove the Crypto-IC from the PCB and send 0x00 to inputs of XOR gates and read the outputs. Better encryption methods were also used (many good encryption methods work in a way to make it difficult to calculate the key from input and output samples-OTP is an exception because the key is as long as the message), but designers had to make compromises because these could introduce too much time delay. Another method was to use 8-bit RAM to **hold the key only** (instead of a hardcoded key in Crypto-IC), which was maintained by a lithium battery. It would be lost if the battery was disconnected.

How could this then be used for Alice's cryptosystem to improve its security? The key idea is that if the system can work only on encrypted program code, then Mallory can't plant a malicious code (or hardware Trojan) in plain, unencrypted form and expect it to work. She needs to know the 8-bit key first. Such a code would simply cause a malfunction that Alice and Bob would notice very easily. Please note that a CPU with a **trap interrupt** would work better here than a Z80 (which doesn't have a trap). It is better to activate a special trap ISR and maintain control when an illegal opcode is detected than to simply go "runaway", which would happen on Z80 in this case. Making or purchasing a reliable and secure ASIC is a big problem for Alice and Bob, but wait for a second. They **don't need it at all!** It is needed for protecting a video gaming machine which will normally "fall in the hands of an enemy" (a pirate trying to copy it). Since Alice and Bob will take care that their crypto-devices don't get captured, they can implement the protection with several standard 74HCxx ICs, like in figure 7.16. They can define the 8-bit (or 24-bit, if they decide to encrypt the address bus as well) keys using simple DIP switches (no microcircuit hard-coding needed), and return them all to the 0x00 setting after each session.

As far as I know, this method was used by the Japanese "TAITO" company for copy-protection. I liked to play their games like "Renegade" and "Double Dragon" in the late 1980s. I don't know if this has ever been tried as a crypto-system security enhancement, but I at least think it is worth a try. Please note that this method doesn't work against a hardware Trojan planted directly inside a CPU since everything inside a standard CPU always runs in "plain" code and data. This is why MyNOR could be a good platform for the implementation of this. A CPU-Trojan can't be planted inside it, simply because **there is no CPU**.

⁶ ASIC - **A**pplication **S**pecific **I**ntegrated **C**ircuit: an IC designed for a specific apparatus. Its silicon die micro-circuits are specifically designed, not its firmware.

7.7 • Asynchronous computers

You probably remember the textbook example of asynchronous and synchronous digital binary counters and their differences. Asynchronous circuits are simple and don't change their binary states in synch with clock pulses. They don't need a clock to work. Binary states in one stage change after all the previous stages which affect it. This is why time delays in asynchronous circuits add up, so they always work slower than synchronous circuits. Synchronous circuits are more complex and they change their binary states in synch with one clock pulse which drives the entire device in synch.

The basic textbook standpoint is that synchronous circuits are more advanced. Well, they are, from the standpoint of speed and functionality. The question is whether the asynchronous circuits would work better from a security standpoint? The answer is yes, because they are less susceptible to **side-channel attacks**, including TEMPEST! We will proceed with an analysis of this, after you take a look at the two following figures, just to repeat the basic principles.

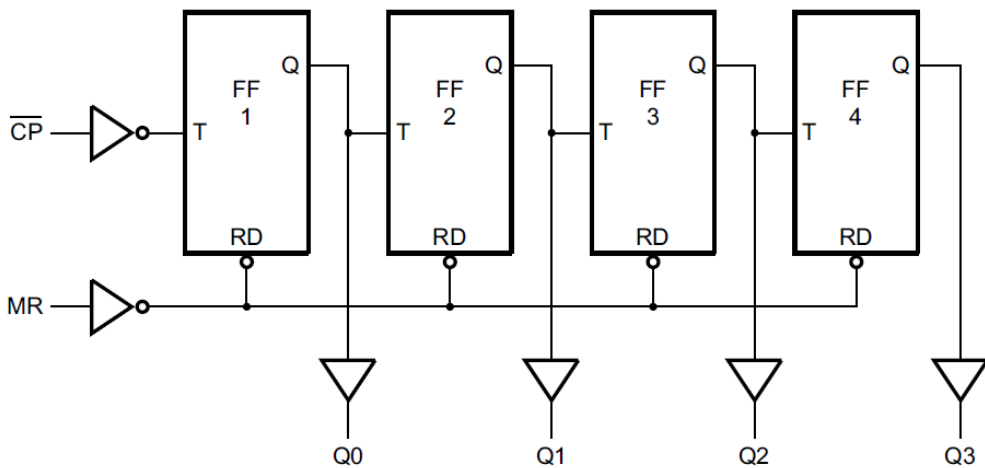


Figure 7.17 - An asynchronous binary counter, like for example the 74HC393

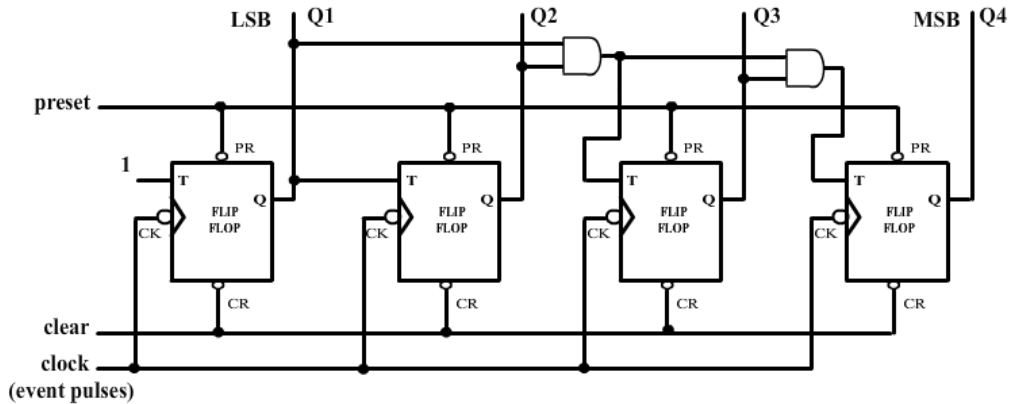


Figure 7.18 - Synchronous binary counter, like for example the 74HC163

T-bistables⁷ of an asynchronous counter (figure 7.17) toggle their state depending only on inputs from previous stages. There is no clock pulse signal. Time delays of single flip-flops add up as a consequence, so a time delay required to flip the most significant bit (Q3) is 4x the delay of a single flip-flop. The synchronous counter (figure 7.18) toggles all the bits in synch with the CK (clock pulse) simultaneously, so the delays don't add up and can work faster. Speaking about this, the circuit in figure 7.18 has a small error, which probably wouldn't affect its operation, maybe only at its maximum speed (maximum CK frequency). Try finding it yourself.

The reason why synchronous devices are more susceptible to TEMPEST attacks is simply that the whole circuit flips all of its bits simultaneously, which creates surges of RF energy on each CK pulse transition, unlike asynchronous devices, where flip-flop transitions are more distributed in time. When monitoring the device's RF signatures during a TEMPEST attack, it is easier to tune to defined frequencies of a synchronous device and observe surges of energy on basic frequency and its higher harmonics (this is the very basic procedure to start a TEMPEST attack, please look at section 6.1 again).

This idea originated in analogue radio-communication long before its possible application in digital computing circuits. When Eve starts hunting for Alice's concealed transmitter, she first needs to tune into her transmitter before she can even start to triangulate her location. Radio transmitters usually transmit the RF energy in a narrow band around the carrier frequency. This is why Eve can observe "spikes" on her RF spectrum analyzer because the RF energy transmitted is concentrated in relatively narrow bands (e.g. 8kHz voice

⁷ T stands for "toggle" - a type of a bistable (or "flip-flop" - "FF") of which latched output toggles its state on each active edge on its T-input in asynchronous variant, or on each active edge on CK input if T=1 in synchronous variant.

bandwidth on a 10MHz AM modulated carrier). **Spread-spectrum**⁸ systems were invented to mitigate this. If the RF energy is distributed along a wider bandwidth, there will be no visible concentrated spikes, and Eve won't even be able to tune into Alice's transmitter signal, let alone start the triangulation procedure.

The same principle is applied when using asynchronous circuits to mitigate TEMPEST attacks. If RF pulses are difficult to detect and isolate, Eve won't be able to detect what is going on inside Alice's computer. Eve needs precise timing for her attack to succeed, which is much easier on synchronous circuits.

Designing a whole asynchronous system from a scratch may be difficult for Alice and Bob. What they can do is to take a standard synchronous computer and modify its clock pulse generator to work with **high jitter** that has been **deliberately introduced**. It needs to be switched to stable synchronous mode only when performing some operations like UART communication. Remember, in section 5.2 (SD-to-cassette copier) we also had an MCU system with a **variable** clock signal, with controlled jitter.

The system will work much slower now, because if for example, an MCU can run on a maximum frequency of 10MHz (meaning that the minimum half-period can't be less than 50ns), the frequency will then be reduced to for example 2MHz, to create enough space for the clock pulses to shift left and right, but always keeping the half-periods higher than 50ns. The pulse shifting can be controlled by a good TRNG. As you can see, its possible applications in crypto-systems are practically endless.

Turning high-jitter mode on and off must be done carefully, with a carefully designed circuit, to avoid too fast transitions (i.e. pulses with half-period less than 50ns), so Alice doesn't end up **clock-glitching**⁹ her computer.

8 One of the simplest is frequency hopping. The RF carrier frequency hops quickly - changes according to the sequence programmed both in Alice's and Bob's transceivers. If they use the 8kHz channel bandwidth and hop between 20 channels, the RF energy will be smeared over a 160kHz span, with much lower peak amplitudes. This is not to be confused with voice scrambling (explained in 4.2.1.2) which is a different procedure, performed at audio frequencies, in time-domain. What is less known is that one of the co-inventors of an early version of frequency hopping method (with a confirmed U.S. patent No. 2,292,387) is Hedwiga Eva Maria Kiesler, better known as Hedy Lamarr. Besides being a famous and beautiful actress, she studied mathematics, mostly on self-taught basis. She claimed that it helped her maintain her mental stability and link with the true, real world. Younger generations could learn a lot from her example.

9 Similar to the aforementioned voltage-glitching attack (lowering the Vdd supply voltage to make a computer perform a faulty action to reveal secret data), the clock-glitching attack is performed by sending a several CPU clock pulses too fast (e.g. 10ns pulses to a CPU designed to operate on 10MHz max) which will also make the CPU under attack perform a faulty action. Typically, the CPU will fetch one instruction from memory, and simply fetch the next instruction (without decoding and executing the previous one, effectively skipping it). If precisely timed, it will make the CPU skip some crucial checks, like e.g. checking a PIN entered to a credit card, and proceed like the correct PIN was entered. Once again, precise timing on an asynchronous device or a device clocked with high random jitter is impossible.

7.8 • DIY device-a supervisor for a “suspicious” commercial device

We already had an example of this method in section 7.3 where an extra ATmega8 MCU was introduced to protect a Funcard from fault injection attacks. The idea is to introduce **a very simple device** (the ATmega8 is much simpler than a “suspicious” PC and runs a much simpler program than a Funcard), which Alice can easily design and program (keeping its design secret) to a much more complex system, which is much easier to attack. Another example is the ZMC Crypto Dev Shield, with its extremely simple supervising circuits: WDT, RAM execution alarm, and ROM-write alarm, which increase security.

This supervisor device can monitor the flow of input data and various voltage levels, to prevent fault injection attacks. It can also monitor suspicious devices (e.g. a cellphone) for RF emanations if Alice suspects that her cellphone is spying on her and transmitting RF when it shouldn’t be.

There are also many other possibilities. The key idea is to protect a large, complex, and vulnerable system, well known to the potential attacker with a very simple, cheap device, of which the details are easy to be kept secret. This can work because suspicious actions (like for example entering badly formatted data, or lowering the supply voltage, or jamming) are usually very easy to detect.

Gross errors (or so-called “blunders”) in large systems operating on a **need-to-know** basis (like 3-letter agencies, but also many privately-owned hi-tech companies trying to preserve their vital secrets) are quite common. I am not saying those people are stupid. They simply have to follow complicated and sometimes very ambiguous security procedures. It is very difficult to properly balance between following procedures to preserve vital secrets and passing all necessary information needed to complete a certain task properly.

I remember two Cold War jokes that illustrate this very well. The events that they describe are too much exaggerated to be true, but the principles behind them inevitably lead to similar errors in practice very often. This will help to explain why the planting of rigged ICs in the East German factory failed.

1. A Soviet agent was thoroughly ready to be sent undercover in Washington. He received several false passports, learned to speak perfect English, prepared a good cover story, and everything else. After travelling using several airplanes, ships, and trains, with a new identity for each voyage, he finally arrived in D.C. He was told to contact Mr. Montgomery, a deep undercover Soviet spy at Elm Street 1643. He entered the apartment block and immediately found an apartment on the ground floor with the tag “MONTGOMERY” on the door. He rings the bell. A man opens the door. He says “Today I met Ms. Daisy and we had a drink together.” (a passphrase). Before slamming the door shut, Mr. Montgomery yelled “Loony spooks! Again! How many times have I told you - I am Montgomery, but I am just a **tailor**, and Montgomery the **spy** lives on the 3rd floor!!”.

Seems impossible? What would you do in place of any of these characters involved? Think carefully! Inform your superiors that somebody is repeatedly making serious security errors because there is also another Montgomery in this building, and **not cause mayhem** within the KGB? Montgomery the tailor doesn't care much about Russian spies and the Cold War in general (except being occasionally annoyed by their surprise visits). Otherwise the FBI would have stormed the building long ago. They haven't done any direct harm to him, after all. Maybe he has already tried to report it, but has simply been ridiculed (this situation may seem too funny even to an average FBI agent), and didn't care to try again. It is also possible that the tailor knows his namesake from the 3rd floor is a spy, but maybe he thinks that he works undercover for the FBI (all of his unannounced visitors spoke perfect English and resembled average Americans). Please note that he called them “**loony spooks**”, and not “**damn reds**” or “**commie scum**”. Maybe it would be easiest to simply advise Montgomery the spy to relocate without informing his superiors about the real reason and causing additional problems for them all. But wait, if Montgomery the tailor knows who he is, why in the world is he still there on the 3rd floor? It is also possible that Montgomery the spy **has never been told** about his colleagues' multiple visits to his namesake on the ground floor?! He probably thinks his agents are well informed to go to the 3rd floor. This would be a normal and adequately precise procedure, right? Nobody involved (especially the visiting KGB agents) want to be embarrassed by this ridiculous situation. So everybody involved, including the tailor are simply keeping to the status quo, and they are “happy” with that.

*If you think this scenario is impossible, please watch “**The Americans**” series and pay attention to what happened after Paige Jennings (a teenage daughter of two Russian spies operating in the USA during the Cold War) tells her pastor in the local church about the true identity of her parents after they told her who they are. Will the pastor break the sacred rule of secrecy of confession? Will he be taken seriously if he goes to the FBI “armed” only with hear-say information received from a confused teenage girl? Her father has already become a good “friend” with his first-door neighbour who is an FBI agent. Even to him, he appears to be a nice average law-abiding American family man. Will Mr. and Mrs. Jennings inform their superiors about their daughter’s “change of heart” and thus put her in great danger? The plan for her had already been made in detail: to be trained as a top-secret spy, a so-called “second generation illegal”, to be infiltrated in the top ranks of FBI, since she was born in the USA. Kill the pastor? Maybe along with his wife, since she also probably already knows about it? It might be difficult to convince Paige they died in a car accident.*

Simply keeping up the status quo might be the most painless solution for everybody, although this doesn’t seem so at first.

2. *An agent was thoroughly ready to be sent undercover to the Soviet Union, the same as his “colleague” from the previous joke. He learned to speak perfect Russian, got false papers, and everything else... And yet, immediately after landing at Moscow Sheremetyevo airport he was arrested by the KGB and sent to jail. Why? The answer is: He was black.*

Too much need-to-know might lead to a ludicrous situation like this, although anyone with a minimum level of common sense knows it is impossible to pass a black man (of African origin) as a Russian. The possible explanations are similar to those in the tailor-and-spy joke, so I will let you work them out. I will give you a hint. What would you do if you learned that somebody from your organisation is training a black operative for a deep-undercover mission in Russia? Maybe he is being framed? Maybe it’s a joke? Does he know exactly where is he being sent? As a legal or an illegal?...

Now it's time to answer why that well-planned planting of the hardware-trojanized chips in East Germany failed. First I need to tell you something about the production of electronic components in the East Bloc. All of its countries use the metric system, so they tried to enforce it everywhere. Although for example Croatia has always used the metric system, it has also always been normal to express a steel pipe diameter in inches, not centimetres (I still ask for 1" or 2" pipe when I go to a hardware store).

The standard pin spacing of electronic components produced in East Bloc was 2,50mm, not 2,54mm (1/10 of an inch) which is normal in the rest of the world. Their factories were always producing both variants of every integrated circuit (one with 2,50mm pitch and another with 2,54mm). The metric versions were intended for their use, while the "imperial" versions were produced for exportation, those included illegal clones of for example the Z80, of course.

Somebody must have forgotten the trojanised ICs must be packed in housing with a 2,50mm pin pitch since it is impossible that an East Bloc military would use the 2,54mm version in devices designed for their use. Although the chips looked perfect, when the technician tried to insert them into 2,50mm DIP sockets, he immediately noticed that they wouldn't fit, because the pitch was wrong. This alone was too suspicious, so he sent the ICs to the lab for detailed inspection.

• Conclusion

So, this is it, and I am sincerely hoping you have found all of this interesting, entertaining, educational and inspiring. Now you can see for yourself how many problems haven't been adequately solved. To start solving security problems properly, acquiring a completely different mindset than the one typical to most in the 21st century is a must.

I must remind you that I can't confirm the authenticity of the historical events related to espionage described in this book. I am no historian. The exact authenticity of events (did it happen like I described, down to the last detail) is **not important** for us, the electronic engineers. What is important is that you can check and analyse the technical problems involved and see for yourself that these events were **logically plausible as described**.

The purpose of security engineering is to (with a fully-developed mindset of a professional paranoid, of course) **foresee** these events **as possible and plausible**, and act accordingly in time to prevent them, not to wait for them to happen. Security engineering in the 21st century will be a tough challenge, as even never seen brand new problems emerge.

I wish you good luck.

• References

- [1] Bruce Schneier, "Secrets and Lies – Digital Security in a Networked World", Wiley Publishing, 2000.
- [2] Bruce Schneier, Neil Ferguson, "Practical Cryptography", Wiley Publishing, 2003.
- [3] Chris Dobson, Ronald Payne, "Dictionary of Espionage", Grafton Books, 1986.
- [4] Kevin Mitnick, "Art of Deception", Wiley Publishing, 2002.
- [5] David Kahn "Codebreakers", Macmillan Publishing, 1979.
- [6] "CRC -ЦИК- Циклический Избыточный Код и Как Его Восстановить" - reversing the CRC control checksum function
https://powerhacker.net/documents/Reverse_Engineering/codebreakers_journal/CBJ-2004-27.pdf
- [7] James E. Gentle "RNG and Monte Carlo Methods", Springer-Verlag New York, 1998.
- [8] William Nitschke "Advanced Z80 Machine Code Programming", Interface Publications, 1985.
- [9] Wim van Eck, "Electromagnetic Radiation from Video Display Units: an Eavesdropping Risk?", 1985.
- [10] Markus Kuhn, "Optical Time-Domain Eavesdropping on CRT Displays", 2002.
<https://www.cl.cam.ac.uk/~mgk25/ieee02-optical.pdf>
- [11] Juraj Bartolić, „Mikrovalna Elektronika“, Graphis Zagreb, 2008.
- [12] Jacques Bergier, „Vohunstvo v Industriji in Znanosti“, Mladinska Knjiga Ljubljana, 1974.
- [13] Peter Gutmann, „Data Remanence in Semiconductor Devices“, 2001.
- [14] "Adding Backdoors at the Chip Level",
https://www.schneier.com/blogarchives/2018/03/adding_backdoor.html
- [15] Sergei Skorobogatov, „Low Temperature Data Remanence in Static RAM“, University of Cambridge, 2002.
- [16] NIST, SP 800-22 Rev. 1a, RNG statistical tests for cryptographic applications
<https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final>

- [17] Luka Matić, Elektor Magazine March & April 2017, "Truly Random Number Generator"
www.elektormagazine.com/150116
- [18] Marco Bucci et al, "A High-Speed Oscillator-Based Truly Random Number Source for Cryptographic Applications on a Smart Card IC" , 2003
- [19] Matthias Wolf , USB TRNG project, Elektor Labs project page
<https://www.elektormagazine.de/labs/usb-random-number-generator>
- [20] Eric S. Raymond, "The New Hacker's Dictionary", MIT Press, 1994.
- [21] Michael Greenwald et al, "Computer Security is Not a Science -but it should be", 2003, University of Pennsylvania
- [22] RSA big semi-prime numbers factoring contest
https://en.wikipedia.org/wiki/RSA_numbers
- [23] Christian Tavernier, Elektor Magazine November 2006, "Smartcards -ABC of Blank Cards for Private Applications"
<https://www.elektormagazine.com/magazine/elektor-200611/18399>
- [24] Crypto AG affairs, "The Gentelman's Agreement"
<https://www.cryptomuseum.com/manuf/crypto/friedman.htm>
- [25] Luka Matić, Elektor Magazine January & February 2019, "One-time Pad Crypto Shield"
<https://www.elektormagazine.com/180543>
- [26] Luka Matić, Elektor Magazine May & June 2020, "Tamper-Evident Box"
<https://www.elektormagazine.com/180445>
- [27] Luka Matić, Elektor Labs project, "Secure SD card-to-SD card Copier"
<https://www.elektormagazine.com/labs/secure-sd-card-2-sd-card-copier>
- [28] Egon Fred Warnke, "Tonbandtechnik ohne Ballast", Franzis-Verlag München, 1969.
- [29] Paul Hetrelezis "Retro Audio – a Good Service Guide", Elektor International Media BV, 2017.
- [30] Luka Matić, Elektor Labs project, "Secure SD card-to-Cassette tape Copier"
<https://www.elektormagazine.com/labs/secure-sd-card-2-cassette-tape-copier>
- [31] Lee Alan Hart, a complete, Arduino-size „ZMC“ Zilog Z80 system
<http://www.sunrise-ev.com/z80.htm>

[32] Luka Matić, Elektor Labs project, „Crypto Dev Shield for ZMC Zilog Z80 system“
<https://www.elektormagazine.com/labs/crypto-dev-shield-for-zmc-zilog-z80-system-1>

[33] Luka Matić, Elektor Labs project, „Magnesium Bulb Analogue Memory add-on for Tamper-evident Box“
<https://www.elektormagazine.com/labs/magnesium-bulb-analogue-memory-add-on-to-the-tamper-evident-box>

[34] “MEGGAFLASH”, an active manufacturer of magnesium flash bulbs
<http://www.meggaflash.com/>

[35] “MyNOR”, a CPU-less computer by Dennis Kuschel
<http://www.mynor.org/>

[36] “TinySA”, a poor man’s 2.8inch pocket spectrum analyser
<https://tinysa.org//>

[37] „TEMPEST for Eliza“, a PC software for shortwave RF TEMPEST
<http://www.erikydy.de/tempest/>

[38] Elektor SDR (shortwave software-defined radio) hands-on start kit, an Arduino Shield
<https://www.elektor.com/elektor-sdr-hands-on-kit>

[39] Elektor SDR kit for Raspberry Pi, USB stick, wide-band up to high UHF range
<https://www.elektor.com/elektor-raspberry-pi-rtl-sdr-kit>

[40] Jon Erickson „Hacking – the Art of Exploitation“, No Starch Press Inc., 2008.

[41] “The THING” ,a passive RF listening device planted to USA embassy in USSR
[https://en.wikipedia.org/wiki/The_Thing_\(listening_device\)](https://en.wikipedia.org/wiki/The_Thing_(listening_device))

[42] “Bitcoin: A Peer-to-Peer Electronic Cash System”, whitepaper, Satoshi Nakamoto, 2009.
<https://bitcoin.org/bitcoin.pdf>

[43] Web pages with more extensive information about Smartcards
<http://www.cardman.com/cards.html>
https://www.weethet.nl/english/smartcards_types.php

[44] BasicCard, a Smartcard with an operating system and a BASIC interpreter
<https://www.zeitcontrol.de/en/products/basiccard>

[45] Dumb phones web page
<https://bestdumbphones.com/>

- [46] Cold boot attack and other tricks by Nina Blum
<https://www.youtube.com/user/elbefuchs>
- [47] Anatol I. Zverev, "Handbook of Filter Synthesis", 1967, Wiley Publishing
- [48] Anatol I. Zverev, H.J. Blinchikoff "Realization of a Filter with Helical Components", 1961, IRE transactions on component parts
- [49] Stefan Mangard, Elisabeth Oswald, Thomas Popp "Power Analysis Attacks: Revealing the Secrets of Smart Cards" ,2007, Springer
- [50] Joseph Weisberg, "The Americans", Cold War spy series, 2013-2018,
<https://www.imdb.com/title/tt2149175/>
- [51] Florian Henckel von Donnersmarck " Das Leben der Anderen ", Cold War spy film, 2006
<https://www.imdb.com/title/tt0405094/>

• Index

Symbols

1-wire 137, 162

9/11 23

62256 72, 135, 176, 177

A

access time 70, 71, 74

acetylene torch 97

acid 26, 32, 65

ADC 40, 140

address bus 39, 70, 71, 133, 135, 170, 171, 206, 207

AES 36, 78, 81, 101, 102, 103, 117, 145

AGC 128, 155, 183, 184, 185, 187

Alan Turing 90

al-Durayhim 33, 81, 94, 99

algorithm 33, 34, 36, 72, 95, 103, 106, 118, 145

al-Khwarizmi 33

Amphenol 201

analogue memory 75, 121, 141, 143

applet 38

arbitrary code 39, 40, 164

Arduino 112, 132, 133, 217, 218

ASCII 15, 53, 100, 103, 151, 155, 162, 201

asymmetric encryption 12, 101

Atmel 39, 40, 55, 84, 108, 135, 196

avalanche 48, 87

AVR 39, 135, 196

B

backdoor 59, 60, 216

ballistic weapons 96

bank-switchable 133

bank-switched 71

base58 104, 193

base64 104

BasicCard 108, 196, 218

bistable 69, 71, 159, 209

bitbanging 112

Bitcoin 94, 104, 106, 191, 192, 193, 218

bit-flipping 74

black box 13, 22, 78

blockchain 61, 94, 192, 193, 194

blueprints 13, 40, 55, 56
 Blum-Blum-Shub generator 83
 BND 111, 181
 brute-force 12, 32, 33, 34, 35, 40, 92, 94, 95, 97, 98, 105, 106
 BSA 87, 88
 buffer-overflow 38, 39, 40, 98, 116, 137, 138, 150, 158, 160, 162, 170, 179
 Buffon 81, 82
 Bühler 111
 bumping 97
 burst transmitter 13
 button-bounce 19, 201, 202

C

Caesar 15, 34, 35, 205
 cavity resonator 188, 189
 chosen-plaintext 90
 ciphertext 21, 26, 32, 34, 35, 52, 53, 90, 91, 92, 94, 101, 103, 201
 CISC 40
 CMOS 59, 76, 140, 205
 CMRR 87
 code-injection 137, 138, 168, 170
 cold-boot 73, 74, 75, 121, 142, 179
 Cold War 10, 15, 16, 19, 21, 22, 23, 44, 47, 56, 91, 96, 110, 117, 181, 204, 212, 213
 communism 57, 110
 control gate 63, 64, 67
 copy-protection 205, 207
 counter-espionage 14, 16
 Covid 117, 201
 CPU-less computer 146, 170, 218
 CRC 67, 108, 109, 216
 cross-link 60
 crosstalk 41, 42, 43
 CRT 41, 44, 45, 47, 49, 52, 99, 100, 216
 Crypto AG 110, 111, 217
 crypto-analysis 31, 32, 33, 34, 91
 crypto-currency 106, 191, 192, 195
 Crypto Dev Shield 139, 158, 159, 162, 211, 218
 CTC 40, 58, 137, 159, 160, 161, 162, 164, 165, 166, 168

D

data bus 39, 55, 70, 126, 135, 140, 163, 171, 173, 176, 205, 206
 dead drop 117
 decapsulating 32
 decapsulation 32, 58, 68, 181
 deconvolution filter 49, 150
 denial of service 98

depletion-mode 65
dictaphone 127
dictatorship 15, 21
dictionary attack 92
differential amplifier 87
dirty marketing 100
Doomsday Preppers 24
DoS attack 98
dot-matrix printer 52, 99, 150
Douglas Adams 95
DRAM 55, 68, 69, 74, 76
drilling attack 144, 145
DS1307 137, 162
dumb terminal 133, 200
dumpster-diving 11, 62, 66, 91
DUT 172
duty cycle 190
dynamite 97, 98

E

eavesdropping 24, 25, 42, 78, 95, 105, 158, 191
EEPROM 28, 40, 41, 55, 57, 59, 64, 65, 71, 74, 108, 137, 148, 170, 196
electro-migration 66, 68
electronic warfare 42
El-Gamal 12, 101
encryption 11, 12, 13, 15, 16, 19, 20, 31, 32, 34, 36, 54, 61, 81, 82, 88, 90, 91, 92, 93, 94, 95, 96, 98, 99, 100, 101, 102, 103, 104, 105, 106, 116, 117, 118, 119, 123, 124, 145, 146, 182, 183, 187, 191, 199, 205, 207
enhancement-mode 64
ENIGMA 90, 91, 101, 145
e-paper 53
espionage 11, 14, 15, 16, 24, 215
Eurochip 107, 108, 109, 111, 145
exclusive or 34

F

fall time 69, 74
FAT32 87, 124
fault injection 32
FCC 52
FERAM 59
ferroelectric 59
Filmnet 100
fingerprint 19, 27
FLASH 27, 28
flicker 41, 95

flip-flop 209
 floating gate 63, 64, 65, 67, 68
 flutter 127, 128
 force the envelope 93
 Fourier 40, 44, 49, 79
 FPGA 56, 57
 frequency-domain 99, 187
 frequency hopping 20, 210
 FS-5000 Harpoon 23
 FSK 116, 127, 128, 140, 183
 Funcard 108, 191, 196, 197, 198, 199, 211

G

Galaksija 131, 132, 133
 GNFS 106
 golden chip 58, 126
 GPS 20, 187
 GSM 16, 17, 116, 145

H

hardwire 56, 59, 121, 137, 144, 146, 190
 Hedy Lamarr 210
 Heisenberg 105
 helical resonator 45, 46, 188
 histogram 80, 87, 88
 honeypot 109
 hot carriers 64, 65, 67, 70, 176
 hot electrons 65
 HP48 71, 72

I

I2C 112, 137, 144, 162
 ICBM 96
 invisible ink 13, 26
 ionosphere 118

J

jitter 84, 210

K

Kevin Mitnick 24, 216
key-logger 12, 26
KGB 96, 212, 213
known-plaintext 35, 90
Kuschel 124, 146, 148, 218
KZU 83

L

LBA 27, 62, 126
ledger 193
Lee Hart 133, 135, 159, 161
Li-ion 24
linear-congruential generator 82
live drop 117
lock picking 97
locksmithing 98
look-up table 104, 140, 141, 169
lossy 116
LTSpice 180
lumped 45, 51, 200

M

machine code 11, 37, 38, 39, 100, 101, 132, 135, 159, 164, 165
machine-code monitor 135
magnetron 188, 189, 190, 191
malware 31, 38, 54
Manhattan Project 92
Matlab 180, 183, 185, 186, 187
Matthias Wolf 88, 217
memory map 39, 159
metadata 27, 28
Mg-bulb 142, 144, 145
Mg-flash 141
microcassette 127, 131
microcode 146
microfilm 26
micro-probing 68
microstrip 51, 52, 188
microwave 10, 29, 43, 47, 50, 51, 118, 149, 150, 188, 189, 191
millennium bug 190
modding 62, 127
modulation index 189
mono-alphabetic cipher 32, 33, 34, 82, 94, 99, 101, 103
Moore's law 14

Morse 15, 19, 20, 41, 54, 95, 201, 202, 203, 204
 mutually prime 101
 MyNOR 9, 124, 146, 147, 148, 170, 207, 218

N

nichrome 147
 NIST test 78, 159
 NMI 137, 138, 140
 NMOS 59, 76, 140
 Non-Aligned Movement 15
 NOP 41, 138, 165, 166, 167

O

obfuscated code 41, 61, 101, 121
 obfuscation 99, 101, 140, 145, 146
 obscurity 145, 146, 207
 off-the-shelf 13, 28
 OLED 10, 49
 one-time pad 33, 36, 91, 94, 101, 118, 121, 147, 187, 205
 one-time password 147
 one-time programmable 147, 170
 opcode 59, 67, 146, 159, 165, 166, 167, 169, 170, 207
 open-hardware 13, 28, 87
 open-software 13
 open-source 22, 28, 32, 87, 145
 operating system 37, 38, 108, 133, 218
 operational amplifier 182, 185

P

pacemaker 189, 190, 191
 payload 39, 140, 164, 165, 166, 167, 168
 payphone 16, 17, 20, 108
 penetration depth 50, 200
 penetration distance 43
 PGP 12, 13, 20, 36, 78, 99, 103, 104, 105, 145, 191, 192, 199
 phonecard 108
 phosphor 47, 48, 49
 phosphorous 47
 photo-detector 48
 photodiode 48
 photomultiplier 48
 phreaking 44, 52, 99, 188
 PI controller 183, 185
 plaintext 12, 15, 26, 31, 32, 33, 34, 35, 36, 41, 50, 52, 53, 55, 60, 82, 90, 91, 92, 94, 100, 101, 102, 103, 116, 118, 123, 131, 155, 193, 194, 199
 poison 23

polysilicon 63, 64
porting 121
POSTE RESTANTE 121
post-processing 88
potassium permanganate 145
power-analysis 97
power-cycle 159, 179
prime number 35, 82, 102, 106
primitive root 82
private key 12, 32, 35, 42, 73, 101, 102, 104, 191, 193, 194, 198
PRNG 36, 78, 81, 82, 83, 84, 103
propagation delay 69, 74
PS/2 52, 155, 156, 157
pseudo-random 41, 52, 82, 83, 108, 119, 159
public key 35, 36, 101, 102, 104, 117, 193, 198
PWM 87, 190, 191
PZT 59

Q

qbit 105
quasar 118, 145, 182
quasi-random 159

R

radio-location 20, 77
raking 97
real programmer 169, 170
real-time clock 61, 137
resonator 45, 46, 157, 188, 189
ring-out 41, 151, 152, 155, 201
rip the envelope 93
RISC 40, 146
runaway code 67, 141, 169

S

safe house 126
Sampoong 18
scrambling 52, 78, 99, 100, 145, 210
semi-prime number 35, 106
session key 36, 95, 102, 103, 104, 117
SHA-1 61, 82, 94, 102, 109
SHA-256 94, 109
shear line 97
shimming 97
Shor's algorithm 34, 106
shortwave 19, 158, 187, 201, 218

shuffling 10, 47, 52, 78
 side-channel 32, 40, 41, 42, 208
 SIGSALY 118, 182, 183, 186, 187, 188
 silicon dioxide 67
 Sinclair 15, 106, 127, 131
 skyscraper 23
 Slim Jim 97
 smashing the stack 38, 162
 snapshot 60
 social engineering 24
 softcore 57
 solvent 26
 spectrum analyser 50, 156, 201, 218
 spread-spectrum 210
 SRAM 39, 40, 55, 59, 60, 61, 66, 68, 69, 70, 71, 72, 73, 74, 75, 76, 88, 119, 121, 135,
 141, 142, 144, 146, 150, 159, 160, 162, 163, 164, 166, 167, 170, 171, 172, 173, 174,
 175, 176, 177, 178, 179
 stack pointer 39, 159, 163, 166, 167
 STASI 16, 26, 57, 89, 123
 state-of-the-art 95
 steganography 100, 145
 subliminal channel 42
 superhet 19
 superheterodyne receiver 19
 switch-mode 155, 200
 symmetric encryption 36, 102, 103, 104

T

tamper 119, 120, 124, 126, 137, 141, 217, 218
 TCM3105 127, 128
 TELEFUNKEN AG 23
 telegraph 19, 20, 42, 54, 77, 95, 123, 201, 202
 telephone 16, 20, 42, 44, 53, 100, 107, 116, 127
 TEMPEST 12, 24, 25, 26, 27, 36, 42, 44, 46, 47, 49, 50, 51, 52, 53, 54, 74, 78, 90, 99,
 100, 109, 116, 121, 133, 150, 152, 155, 156, 157, 158, 176, 179, 180, 200, 208, 209,
 210, 218
 thermal lance 97, 99
 thermite 23, 97
 The THING 218
 threshold voltage 69
 time-domain 99, 100, 187, 210
 time-stamping 121
 timing attack 32, 40, 42, 97
 TinySA 218
 TraNOR 148
 transmission line 42, 51

trench warfare 42, 90

TRF 19

triangulation 19, 210

TRNG 78, 79, 81, 82, 83, 84, 86, 87, 88, 93, 124, 126, 198, 200, 201, 202, 210, 217

true random 13, 19, 36, 54, 78, 83, 106

tuned radio frequency receiver 19

turntable 182

typebar 40

typewriter 26, 40, 42, 116, 200

U

UART 112, 116, 121, 128, 137, 155, 158, 159, 160, 161, 162, 164, 165, 166, 168, 194, 198, 210

UHF 44, 45, 46, 47, 121, 131, 158, 188, 218

UNO R4 112, 114, 115

V

vacuum tube 42, 44, 48, 56

van Eck 12, 44, 46, 47, 52, 99, 157, 188, 216

VENONA 91, 92, 99

Vernam 15, 33, 112

VHDL 57

Vigenere 15

VOIP 17, 75, 187

Voja Antonić 131

von Neumann 39, 40, 67, 135, 137, 150, 158

W

warez 109

Warsaw Pact 15, 22, 23, 111

watchdog 138, 158

wavelength 41, 43, 45, 48, 158, 189

WDT 40, 138, 158, 159, 211

wear-levelling 62

Wichmann-Hill generator 83

WinHex 80

WinZip 80, 81

write amplification 28

X

XOR 15, 88, 92, 108, 109, 126, 146, 205, 206, 207

Y

Y2k 190

Z

Z80 28, 37, 39, 40, 56, 57, 58, 59, 60, 67, 88, 100, 116, 121, 126, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 146, 150, 158, 159, 162, 163, 164, 165, 166, 169, 170, 182, 198, 200, 205, 207, 214, 216, 217, 218

Z-cash 94

ZeitControl 108, 196

Zener diode 85

zeroisation 28, 62, 68, 74, 75, 141

Zilog 28, 39, 40, 56, 58, 67, 100, 116, 131, 217, 218

Zimmermann 12, 103, 104

ZMC 9, 35, 132, 133, 134, 136, 139, 146, 158, 159, 162, 163, 164, 169, 211, 217, 218

A Handbook on DIY Electronic Security and Espionage

Nowadays, security problems are rarely properly solved or correctly addressed. Electronic security is only part of the chain in making a system secure. Electronic security is usually addressed as network or software security, neglecting other aspects, but the chain is only as strong as its weakest link.

This book is about electronic hardware security, with an emphasis on problems that you can solve on a shoestring DIY budget. It deals mostly with secure communications, cryptosystems, and espionage. You will quickly appreciate that you can't simply buy a trustworthy and reliable cryptosystem off the shelf. You will then realise that this applies equally to individuals, corporations, and governments.

If you want to increase your electronic security awareness in a world already overcrowded with networks of microphones and cameras, this is a book for you. Furthermore, if you want to do something DIY by designing and expanding upon simple electronic systems, please continue reading. Some of the devices described are already published as projects in the Elektor magazine. Some are still ideas yet to be worked out.

Complexity is the main enemy of security, so we'll try to keep to simple systems. Every chapter will analyse real-life espionage events or at least several hypothetical scenarios that will hopefully spark your imagination. The final goal is to build a security-conscious mindset (or "to get into a head of a spy") which is necessary to recognise possible threats beforehand, to design a truly secure system.

Don't bother reading if:

- › you think you and your secrets are 100% safe and secure
- › you think somebody else can effectively handle your security
- › you think conspiracy theories only exist in theory –Telefunken's masterpiece the "FS-5000 Harpoon" was built on one!



Luka Matic was born in Rijeka, Croatia in 1976. After graduating from the Automation department of FER Zagreb, Luka started to design secure crypto electronics in cooperation with Elektor. He also gained valuable electronic and physical security experience while working in offshore construction and oil drilling. He now works as a researcher at FER Zagreb, where he hopes to obtain a Ph.D. in secure crypto electronics. Hobbies of Luka are sports, movies, reading, and his beloved cat Toxy.

Elektor International Media BV
www.elektor.com